Optimization Methods and Software
Vol. 00, No. 00, Month 200x, 1–14

RESEARCH ARTICLE

Dualize it: software for automatic primal and dual conversions of conic programs

Johan Löfberg*

Division of Automatic Control, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden

(*Received 14 March 2007; final version received 11 June 2008*)

Many optimization problems gain from being interpreted and solved in either primal or dual form. For a user with a particular application, one of these forms is usually much more natural to use, but this is not always the most efficient one. This paper presents an implementation in the optimization modelling tool YALMIP that allows the user to define conic optimization problems in a preferred format, and then automatically derive a symbolic YALMIP model of the dual of this problem, solve the dual, and recover original variables. Applications in flexible generation of sum-of-squares programs, and efficient formulations of large-scale experiment design problems are used as illustrative examples.

Keywords: Optimization; Conic programming; Modelling software

1. Introduction

The purpose of a modelling language for optimization problems is to allow the user to define a problem in a convenient symbolic format. The modelling tool will then extract numerical data, solve the problem using either an internal or external low-level solver, and return the solution in a user-friendly way. The problem addressed in this paper is the case when the optimization problem is not uniquely defined from the symbolic problem definition, due to primal and dual model interpretations, hence leading to a degree of freedom when deciding how to interpret and solve the problem.

This paper describes how the MATLAB^{TM}based modelling language YALMIP [4] deals with these issues in the context of conic programming [1], and proposes a feature which allows the user to concentrate on application modelling, while the software takes care of primal-dual interpretations and conversions.

In addition to describing the software feature and its implementation, an important goal of this paper is to communicate the crucial fact that how you model and interpret a conic optimization problem can make a huge difference, in terms of computational complexity. A fact which should be obvious, but by simply judging from discussions with users of YALMIP, there is a lot of confusion among casual users of conic optimization regarding the difference between primal and dual models and forms, and the difference between high-level symbolic descriptions of a conic program and the numerical data that actually define the problem, and thus complexity. It is not uncommon to see academic reports and presentations where

*Email: johanl@isy.liu.se

ISSN: 1055-6788 print/ISSN 1029-4937 online

DOI: 10.1080/1055678xxxxxxxxxxxxx
http://www.informaworld.com

the model and the reported CPU time to solve the problem simply do not match, indicating that unnecessarily complex representations of the models were used.

The paper starts in Section 2 with a brief introduction to conic programming and the general notation used in the paper. Section 3 illustrates the underlying problem with a number of small motivating examples. The implementation is outlined in Section 4, followed by extensions and some discussion in Section 5. Applications and examples are reported in Section 6.

2. Conic programming

The paper revolves around modelling of primal and dual real-valued conic optimization problems [1].

$$\mathcal{P}: \ \underset{X}{\text{minimize}}\ C \bullet X \text{ subject to } \mathcal{A}(X) = b,\ X \in \mathcal{K} \tag{1}$$

$$\mathcal{D}: \ \underset{y}{\text{maximize}}\ b^T y \text{ subject to } C - \mathcal{A}^T(y) \in \mathcal{K} \tag{2}$$

For readers unfamiliar with these definitions, they might seem a bit abstract, but they are simply a notational form used to unify linear programming (LP), second order cone programming (SOCP) and semidefinite programming (SDP). In linear programming, $\mathcal{K}$ is the positive orthant $\mathcal{K}_L = \{X \in \mathbf{R}^m \mid X \geq 0\}$, the second order cone (sometimes called the quadratic cone or the Lorentz cone) is given by $\mathcal{K}_Q = \{X \in \mathbf{R}^m \mid X = \begin{bmatrix} x_1, x_2 \end{bmatrix}^T,\ x_1 \in \mathbf{R},\ x_2 \in \mathbf{R}^{m-1}, ||x_2|| \leq x_1\}$ while the semidefinite cone is the set of real symmetric positive semidefinite matrices $\mathcal{K}_S = \{X \in \mathbf{S}^m \mid X \succeq 0\}$. The expression $C \bullet X$ denotes the inner product, $\text{trace}(C^T X)$. For all problem classes, $y \in \mathbf{R}^m$. The linear operator $\mathcal{A}$, the adjoint operator $\mathcal{A}^T$, C and b define the specific problem instance data.

$$\begin{aligned}
\mathcal{LP}: &\quad \mathcal{A}(X) = AX,\ \mathcal{A}^T(y) = A^T y,\ A \in \mathbf{R}^{n \times m},\ C \in \mathbf{R}^m,\ b \in \mathbf{R}^n \\
\mathcal{SOCP}: &\quad \mathcal{A}(X) = AX,\ \mathcal{A}^T(y) = A^T y,\ A \in \mathbf{R}^{n \times m},\ C \in \mathbf{R}^m,\ b \in \mathbf{R}^n \\
\mathcal{SDP}: &\quad \mathcal{A}(X) = \begin{bmatrix} A_1 \bullet X \\ \vdots \\ A_n \bullet X \end{bmatrix},\ \mathcal{A}^T(y) = \textstyle\sum_{i=1}^n A_i y_i,\ A_i \in \mathbf{S}^m,\ C \in \mathbf{S}^m,\ b \in \mathbf{R}^n
\end{aligned}$$

The problems addressed in this paper allow mixed cones $\mathcal{K} = \mathcal{K}_L \times \mathcal{K}_Q \times \mathcal{K}_S$, and problems with multiple instances of each cone.

2.1. *Free variables and equality constraints*

It is convenient to extend the definitions of the primal and dual forms slightly to allow free variables in the primal form, corresponding to equalities in the dual.

$$\mathcal{P}': \ \underset{X,t}{\text{minimize}}\ C \bullet X + f^T t \text{ subject to } \mathcal{A}(X) + F^T t = b,\ X \in \mathcal{K} \tag{3}$$

$$\mathcal{D}': \ \underset{y}{\text{maximize}}\ b^T y \text{ subject to } C - \mathcal{A}^T(y) \in \mathcal{K},\ Fy = f \tag{4}$$

These models are not, in theory, more general than the standard models $\mathcal{P}$ and $\mathcal{D}$, but simplify modelling and notation since they avoid introduction of difference

of positive variables (to model free variables in the primal form) and double sided inequality constraints (to model equality constraints in the dual form).

3. Dualization

A modelling language for conic optimization problems allows the user to define a problem in a convenient symbolic form, extracts the numerical data $(C, \mathcal{A}, b, F, f)$, solves the problem, and returns the desired solution variables. The problem addressed in this paper is the case when the numerical data is not uniquely defined from the symbolic problem definition. This is always the case since it is impossible for the software to know whether the user intended to model the problem in a primal form $\mathcal{P}'$ or dual form $\mathcal{D}'$.[1]

It is easy to initially be fooled by the notion of "primal-dual solvers" and believe that it makes no difference how a problem is modeled, in terms of primal and dual notation. This is however not the case. The problem data (C, A, b, F, f) is not only non-unique, but more importantly, can have completely different sizes depending on how a problem is interpreted.

To motivate the development of software based support for dealing with primal and dual versions of a model, we start with some motivating examples.

Example 3.1 Consider the following trivial LP

$$\begin{array}{ll} \text{minimize} & z_1 \\ \text{subject to} & z_1 + z_2 + z_3 = 1 \\ & z_1 \geq 0 \\ & z_2 \geq 0 \\ & z_3 \geq 0 \end{array} \tag{5}$$

From the definitions of primal and dual forms, a problem in primal form is trivially detected.

$$c = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T, A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, b = 1$$

However, the problem can also be interpreted in dual form with an equality.

$$c = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, A^T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, b = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, F = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, f = 1$$

■

The ambiguity in the problem definition is the center of attention in this paper. For the person defining the problem, assuming solid knowledge of conic optimization and the way problems are solved, it is probably clear that the LP above is stated and should be interpreted in primal form, since it involves a simple cone variable $z \geq 0$ and one equality constraint. If the modelling tool has primal form as the assumed format, the primal structure will be used. However, if the dual format is assumed by the modelling tool, the more computationally demanding dual form of the problem will be extracted.

[1]Note that we talk about primal and *forms*, not primal and dual problems. The problem declared by the user is the primal problem, but it can be given and interpreted in either primal or dual form.

Solving the "wrong" version of the LP described above would not pose any problems due to the small size, so let us consider a more realistic and devastating example where detecting and solving the correct version is crucial.

Example 3.2 Consider the binary quadratic optimization $\min_{x\in\{-1,1\}^n} x^TQx$ arising in, e.g., MAXCUT computations [3]. A simple semidefinite relaxation of this problem is given by

$$\begin{array}{l}\text{minimize } \ Q \bullet X \\ \text{subject to } X_{ii} = 1,\ i = 1,\ldots,n,\ X \in \mathcal{K}_S\end{array} \tag{6}$$

This can immediately, by manual inspection, be identified as a simple semidefinite programming problem in primal form (e_i denotes the ith unit vector).

$$C = Q,\ b_i = 1,\ A_i = e_ie_i^T,\ i = 1,\ldots,n$$

By explicitly parameterizing the symmetric matrix X with $n(n+1)/2$ variables,

$$X = \sum_{1\le i\le j\le n} E_{ij}x_{ij}, \quad E_{ij} = \frac{e_ie_j^T + e_je_i^T}{2}$$

the problem can alternatively be interpreted in dual form with equality constraints (never mind the details, the interesting feature is that there are $O(n^2)$ variables)

$$C = 0,\ b_k = \begin{cases} -Q_{ij} & \text{if } i = j \\ -Q_{ij} - Q_{ji} & \text{otherwise}\end{cases},\ A_k = -E_{ij},\ 1 \le i \le j \le n$$

$$F_{ij} = \begin{cases} 1 \text{ if } j = i(n+1) - n \\ 0 \text{ otherwise}\end{cases},\ f_i = 1,\ i = 1,\ldots,n$$

Obviously, a much larger and terribly inefficient model. ■

In the last example, it is not as easy as in the first example to automatically detect the preferred primal form version of the problem. To do this, the modelling language has to keep the aggregation of variables, i.e. X has to be treated as a matrix, and not as a matrix composed of scalar variables.

The dualization procedure has been developed to be used in the MATLAB toolbox YALMIP [4]. Models in YALMIP are always interpreted in a disaggregated dual form, hence the efficient primal form would be lost.

It should be mentioned that it is possible to define a problem so that the desired form is respected in the computations. For a modelling tool assuming a dual form, we can manually declare the dual of our problem, solve this problem, and recover the original variables from the duals in the dual problem. Although this is a possible approach for avoiding the problems with the assumed dual form, it severely complicates modelling for the user, especially when there are several cone constraints, free variables and mixed cones. Manual derivation of a dual problem does not scale well for large complex problems in an engineering scenario where rapid prototyping and experimentation is needed.

At this point, one might wonder why it was decided to have a dual form preference in the modelling language to begin with. The reason is two-fold. First, symbolic manipulations simplify substantially when variables are disaggregated. Secondly, a large number of problems, probably most problems in many fields, are most

naturally and efficiently modeled in the dual form. Consider for instance a typical semidefinite programming problem from control theory.

Example 3.3 The goal is to compute the induced $\mathcal{L}_2$ gain of a linear system $\dot{x} = Ax + Bu,\ y = Cx$ with $x \in \mathbf{R}^n$ and $u \in \mathbf{R}^m$. One way to do this is to solve the following SDP [2].

$$\begin{aligned} &\text{minimize} \quad t \\ &\text{subject to} \begin{pmatrix} A^TP + PA & PB \\ B^TP & C^TC - tI \end{pmatrix} \preceq 0,\ P \succeq 0 \end{aligned} \tag{7}$$

To model this in a dual form, we parameterize $P \in \mathbf{S}^n$ and end up with a problem with $1 + n(n+1)/2$ variables and 2 semidefiniteness constraints. If we insist on modelling this in a primal form (which is not unreasonable since P defines a simple primal cone), we are forced to introduce a slack variable S and obtain (for notational simplicity, exploit the fact that t is positive)

$$\begin{aligned} &\text{minimize} \quad t \\ &\text{subject to } S = \begin{pmatrix} A^TP + PA & PB \\ B^TP & C^TC - tI \end{pmatrix},\ (P, S, t) \in \mathcal{K}_S \end{aligned} \tag{8}$$

This problem has $(n+m)(n+m-1)/2$ equality constraints in the primal form, corresponding to $(n+m)(n+m-1)/2$ variables in the dual. This should be compared to $1 + n(n+1)/2$ variables needed if we used the dual form model instead. ■

The example is actually not a particularly bad case for a dualization, but it is already clearly unnecessary to dualize it. A much worse scenario is a problem with a dual form semidefinite constraint in $\mathbf{S}^m$ with only $O(m)$ or less variables. Dualizing this requires introduction of a slack matrix and $O(m^2)$ equality constraints, corresponding to $O(m^2)$ variables in dual form, to be compared with the original $O(m)$ variables.

4. Implementation

In this section, the basic ideas in the implementation of the dualization procedure will be outlined and discussed. To avoid detailed implementation issues, a lot of details, special cases and generalizations will be omitted or only briefly discussed. To begin with, we state the main part of the implementation in pseudo-code form (a more detailed description is given on the next page).

Input Model $\mathcal{F}$ with constraints $\mathcal{F}_1, \ldots, \mathcal{F}_N$
Initialize list of primal cone variables $X = ()$
Initialize list of translation offset $H = ()$
1. for $\mathcal{K} = \{\mathcal{K}_S, \mathcal{K}_Q, \mathcal{K}_L\}$
  for all $\mathcal{F}_i$ **in** $\mathcal{F}$
    if $\mathcal{F}_i$ **defines primal** $\mathcal{K}$**-cone constraint**
      Append primal cone variable in $\mathcal{F}_i$ to X
      Append 0 to H
      Remove $\mathcal{F}_i$ from $\mathcal{F}$
    end if
  end for
end for

2. for $\mathcal{K} = \{\mathcal{K}_S, \mathcal{K}_Q, \mathcal{K}_L\}$
  for all $\mathcal{F}_i$ **in** $\mathcal{F}$
    if $\mathcal{F}_i$ **defines translated** $\mathcal{K}$**-primal cone constraints**
      if variable in $\mathcal{F}_i$ **not part of** X
        Append primal cone variable variable in $\mathcal{F}_i$ to X
        Append offset term in $\mathcal{F}_i$ to H
        Remove $\mathcal{F}_i$ from $\mathcal{F}$
      end if
    end if
  end for
end for
3. for all $\mathcal{F}_i$ **in** $\mathcal{F}$
  if $\mathcal{F}_i$ **defines dual cone constraint**
    Define slack S and append to X
    Append 0 to H
    Append slack equality constraints to $\mathcal{F}$
    Remove $\mathcal{F}_i$ from $\mathcal{F}$
  end if
end for
4. Compile numerical data from remaining primal form model
5. Create symbolic dual model $\mathcal{F}'$ **and dual objective**
Return Dualized model $\mathcal{F}'$, dual objective, primal cones X and offset H

The first crucial step in the dualization is to detect constraints that define what we call primal cone variables, i.e. constraints of the type $X \in \mathcal{K}$. Since the input to the algorithm is a model in dual form, primal cone constraints can be detected by analyzing C and $\mathcal{A}$ in a constraint $C - \mathcal{A}^T(y) \in \mathcal{K}$. LP and SOCP cones are detected trivially if $C = 0$ and $A = -I$ while an SDP cone requires $C = 0$, $\mathcal{A}^T(y) = -\left[\text{vec}(E_1), \ldots, \text{vec}(E_{n(n+1)/2})\right] y$. After detecting constraints intended to define primal cone variables, the associated cone constraints are removed from the list of constraints. Notice that it is important to detect the primal cone variables in a certain order, starting with SDP cones, then SOCP cones, and finally LP cones. The reason is that there may be constraints such as $X \in \mathcal{K}_S, X_{12} \in \mathcal{K}_L$. This should be categorized as a primal SDP cone, the variable in X_{12} should not be declared as an LP cone variable. Instead, the constraint on X_{12} will be dealt with as a dual form constraint in the third step of the algorithm.

A second step involves finding translated primal cone constraints $X - H \in \mathcal{K}$ where H is constant (and symmetric in the SDP case). A trivial variable change will later allow us to define primal cone variables based on these constraint. Finding these variables are just as simple as finding the simple primal cone variables. Note that some care has to be taken during this step. First, variables already defined as primal cone variables, should not be added again. Secondly, for problems with redundant constraints, such as $X \geq 1, X \geq 2$, only one of the constraints can be used to define the translated primal cone variable. Preferably, the most binding constraint should be used and the redundant constraints be removed.

The problems analyzed do not have to describe a purely primal form, but allow also mixed primal-dual forms. To deal with this, the next step is to detect general dual form constraints (i.e. all remaining cone constraints), define slack variables and change the dual form constraint to a set of equality constraints and a new primal cone variable. The dual form constraints are removed from the problem and the slacks are added to the list of primal cone variables.

The remaining constraints should now all be equality constraints, containing both the original equality constraints defined by the user, and equality constraints

defined by the algorithm when slack variables were introduced to deal with dual form constraints. Variables in these equality constraints, not part of any of the primal cone variables, are free variables in the primal form.

From the equality constraints remaining in $\mathcal{F}$ and the definition of primal cone variables and free variables, it is straightforward to extract the numerical data for the (almost) primal form problem $\min\{C \bullet X + f^T t \mid \mathcal{A}(X) + F^T t = b,\ X - H \in \mathcal{K}\}$. Defining the new variable $Z = X - H$ leads to the final primal form model $\min\{C \bullet Z + f^T t \mid \mathcal{A}(Z) + F^T t = b - \mathcal{A}(H),\ Z \in \mathcal{K}\}$. Based on this numerical data, a dual model $\max\{(b - \mathcal{A}(\mathcal{H}))^T y \mid C - \mathcal{A}^T y \in \mathcal{K},\ Fy = f\}$ is created (in symbolic form) and returned to the user.

4.1. *Solution recovery*

The result from the dualization procedure is a new model which can be manipulated as any other model in YALMIP. By solving it, an optimal primal solution y is computed (primal solution in the sense that it is the variable used in the definition of the problem that has been solved). The original variables defining the original problem are however not available, hence they have to be recovered. To do this, it is assumed that a primal-dual solver is used, and that the dual variables are available (dual variables in the sense duals for the constraints in the dualized model). By associating the constraints in the dualized model $\mathcal{F}'$ with the original primal form variables, the variables Z and t are recovered by extracting the duals computed when solving the dualized problem. The original variables X are finally computed from the relation $Z = X - H$.

5. Extensions and discussion

The algorithm described in this paper is rather general and deals with the important linear, second order and semidefinite cone. Still, improvements are possible.

5.1. *Improved cone detection*

At the moment, the implementation only detects primal cones of the form $X \in \mathcal{K}$ (and the translated cone $X - H \in \mathcal{K}$) An extension to mirrored cones $-X \in \mathcal{K}$ is straightforward and will be implemented in a future release. An even more general form that easily can be dealt with is cone definitions of the form $X - H(t) \in \mathcal{K}$ where $H(t)$ is linear in t. This form arise fairly often, with a prime example being modelling of the largest eigenvalue of the matrix X. The problem can easily be recast to a standard primal form, and thus be detected and treated in the most suitable manner in the proposed framework.

5.2. *Convex quadratic problems*

The standard conic form used in this paper does not allow convex quadratic objective functions. Extending the dualization to this is possible, and would improve the modelling capabilities significantly. To dualize problems with a convex quadratic objective in the current implementation, an epigraph formulation with a second order cone is needed. If the original problem is a quadratic program, reformulating the problem as a second order cone problem just in order to be able to perform a dualization would probably lead to an unnecessarily complex model. Unfortunately, there are currently no public implementations of conic optimization solvers with

quadratic terms in the objective, although research implementations have been reported [7]. Hence, support in YALMIP for this problem class is currently not of interest.

5.3. *Logarithmic barrier terms*

Problems with logarithmic barrier terms are common in practice [9]. Many of these models can be dealt with in YALMIP by using a geometric mean objective instead[1], and dualizing this new model. However, this model is unnecessarily complex and introduces a significant amount of new variables and constraints, often making the dualization less beneficial. Conic problems with logarithmic barrier terms in the objective have a duality theory which is very close to the standard linear conic problem. In a highly simplified form, the primal-dual pair for conic programs with logarithmic barriers is given by

$$\mathcal{P}: \ \underset{X}{\text{minimize}}\ C \bullet X - \text{logdet}(X) \text{ subject to } \mathcal{A}(X) = b,\ X \in \mathcal{K} \tag{9}$$

$$\mathcal{D}: \ \underset{y}{\text{maximize}}\ b^T y + \text{logdet}(C - \mathcal{A}^T(y)) \text{ subject to } C - \mathcal{A}^T(y) \in \mathcal{K} \tag{10}$$

Note that the precise definition of the logarithmic barrier *logdet* on the three different cones have been left out from the description here. Additionally, a number of constant terms have been omitted. The interested reader is referred to [8] for the details.

Extending the ideas presented in Section 4 to support logarithmic barrier terms in the objective is fairly straightforward, and the implementation in YALMIP already supports this extension. Details on the primal and dual forms of the conic problem with logarithmic barrier terms are omitted from this paper, in order to focus on the basic idea. An example where logarithmic terms occur is however reported in Example (6.4).

5.4. *Complexity estimation*

It should be emphasized that dualization is not universally applicable, in fact most problems do not gain from a dualization. As a rule of thumb, dualization is beneficial when you have many or large primal cone constraints $X \in \mathcal{K}$, equality constraints, and few dual form constraints.

A useful feature would be to automatically estimate the complexity of solving the given dual form problem, versus the complexity of deriving and solving the primal form. Based on this, an automatic decision whether to dualize or not can be made and guide the user. The argument against such a feature is the possible unpredictability of the software, and the overhead to perform this analysis, which could be significant when repeatedly solving many small problem (a common situation reported by many users of YALMIP)

5.5. *Primalize it*

For completeness, a related feature called `primalize` has been implemented. This algorithm takes a symbolic model as input and derives a symbolic model in primal

[1]YALMIP automatically converts geometric means of eigenvalues to a standard conic problem, using the nonlinear operator framework in YALMIP.

form. The implementation of this algorithm is much easier than the dualization, since it does not require the detection of any aggregate primal cone variables. For each cone constraint $C_i - \mathcal{A}_i^T(y)$, a symbolic primal cone variable X_i is defined and the constraint $C_i - \mathcal{A}_i^T(y) = X_i$ is added to the primal equality constraints. The final result is a YALMIP model `[A*X+F'*t==b, X>0]`. The details will not be discussed further here since the functionality seldom is of any use. An example where a primalization can be useful is however presented in the YALMIP manual.

6. Examples and Applications

The main motivation for the development of the dualization support in YALMIP is two-fold. The first reason is to cater a need in, e.g., the combinatorial optimization community, where SDP problems often are derived and written in a primal form (in contrast to the control community where the dual form seems to be most common). The other main reason was to simplify the development of a highly flexible and general sum-of-squares module in YALMIP [5].

All computations in this section were performed using SDPT3 [8] on a 2GHz computer.

6.1. *Sum-of-squares*

The problem addressed in sum-of-squares, in a simplified setting, is to check non-negativity of a polynomial $f(x)$, or more generally, to find a set of variables t, linearly parameterizing the coefficients of a polynomial, such that global non-negativity is guaranteed in x.

$$f(x) = f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{N} c_i(t) \prod_{j=1}^{n} x_j^{p_{ji}} = \sum_{i=1}^{N} c_i(t) x^{p_i} \geq 0 \ \forall x$$

The idea in sum-of-squares is to replace non-negativity with the, obviously sufficient, condition that the polynomial is a sum of squared terms. Details regarding sum-of-squares will not be discussed here. Readers unfamiliar with sum-of-squares decompositions are referred to [6] for an introduction.

$$f(x) = \sum_{i=1}^{M} h_i^2(x) = \sum_{i=1}^{M} (q_i^T v(x))^2 = v^T(x) X v(x), \ X \in \mathcal{K}_S$$

Hence, if we can find a vector of monomials $v(x)$ and a positive semidefinite matrix X, non-negativity is guaranteed. For a given $v(x)$, equivalence between the two polynomials implies a set of linear equalities on the elements of the matrix X. This together with the semidefiniteness constraint turns the sum-of-squares problem into semidefinite programming problem.

$$\mathcal{A}(X) + F^T t = b, \ X \in \mathcal{K}_S$$

Example 6.1 Consider $p(x) = 1 + 2x^3 + (4t + 5)x^4$. A sufficient basis for a sum-of-squares decomposition is $v(x) = \left[1, x, x^2\right]^T$. Matching of coefficients yields the

following data.

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, A_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, A_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix},$$

$$A_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} A_5 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, F^T = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -4 \end{bmatrix}$$

$$b = \begin{bmatrix} 1, 0, 0, 2, 5 \end{bmatrix}^T$$

■

Deriving the data $\mathcal{A}$, b and F, given the coefficients of the polynomial $f(x)$ is straightforward, and is done automatically in YALMIP. The result from this step is a model intended to be interpreted in primal form. In order to be as general as possible in YALMIP, the model is defined in a symbolic YALMIP format, i.e. the result from the first step in the sum-of-squares module is a YALMIP model of the type `[A*X(:)+F'*t == b, X>0]`. This allows extremely simple addition of additional user-specified constraints on the parametric variables t. To add additional constraints on t, we simply append these to the constraints derived for the decomposition. In principle, it would be possible to solve the problem at this point, but since the matrix X often is large and a major part of the model is given in a primal form, a dualization is typically beneficial[1]. The sum-of-squares module in YALMIP can be summarized with the following steps.

Input Polynomial $f(x,t)$, parametric constraints $\mathcal{F}_t(t)$ and objective $h(t)$
1. Compile primal form SDP constraints $\mathcal{F}_{SOS}(X,t)$from $p(x,t)$
2. Append constraints $\mathcal{F}(X,t) = \mathcal{F}_{SOS}(X,t)+\mathcal{F}_t(t)$
3. Dualize $\mathcal{F}(X,t)$ **and** $h(t)$ **to obtain dual model** $\mathcal{F}'(y)$ **and** $h'(y)$
4. Solve dualized model using any primal-dual solver
5. Recover t **and** X **from dual of dualized model**
Return Optimal X and t

It should be emphasized that since the dualization supports linear, second order and semidefinite cones, the parametric constraints can be defined using all these constructs. Moreover, the problem definitions above are highly simplified in order to make the main ideas clearer. The sum-of-squares module supports various generalizations such as multiple sum-of-squares constraints, matrix sum-of-squares, automatic block-diagonalization and various pre and post-processing routines [5]. This cause no conceptual problems for the dualization approach though, since these extensions merely gives multiple cones and larger problems to be dualized.

[1] An alternative is to explicitly solve for the equality constraints, i.e. derive an image representation of X. This is supported in YALMIP and is controlled with the option `sos.model`. For some advanced sum-of-squares problems, a dualization is not even possible (nonlinear parameterizations, integrality constraints on parametric variables etc) so an image representation is derived by default. A third alternative is to simply solve the problem in the stated form, will a full parameterization of X and a large number of equality constraints in dual form. This is however almost never recommended.

Example 6.2 Consider the nonlinear dynamical system $\dot{x} = f(x)$,

$$\dot{x}_1 = -x_1 - x_1^3 + x_2$$
$$\dot{x}_2 = -x_2$$

To prove stability, we employ a quadratic Lyapunov function $V(x) = x^T P x$. Asymptotic stability is guaranteed if $V(x) > 0$ and $\dot{V}(x) = \frac{\partial V(x)}{\partial x} f(x) < 0$. By replacing the negativity and positivity constraints with sum-of-squares constraint, we can search for a feasible P using semidefinite programming. Setting up this problem in YALMIP is straightforward (for simplicity, we work with non-strict inequalities and disregard the problems associated to deriving strict conditions).

```
>> x = sdpvar(2,1);
>> f = [-x(1)-x(1)^3+x(2);-x(2)];
>> P = sdpvar(2,2);
>> V = x'*P*x;
>> dVdx = jacobian(V,x);
>> dVdt = dVdx*f;
>> constraints = [sos(-dVdt), sos(V)];
>> parametricVariables = P;
>> solvesos(constraints,[],[],parametricVariables);
```

At this point, we realize that the model is unnecessarily complex. The resulting SDP problem will have two SDP constraints (one for each sum-of-squares constraint) and three free variables (from P). Instead, we notice that the sum-of-squares constraint on $V(x)$ is equivalent to $P \succeq 0$. We might just as well add this as a parametric constraints. The net result is that the dualization procedure will mark P as a primal cone variable, thus avoiding to introduce any free variables.

```
>> constraints = [sos(-dVdt), P > 0];
>> solvesos(constraints,[],[],parametricVariables);
```

Adding advanced constraints on the parametric variables is trivial due to the full integration of the sum-of-squares module with the rest of the YALMIP framework, and the generality of the dualization procedure.

```
>> constraints = [sos(-dVdt), P > 0, norm(P,1) < 1];
>> solvesos(constraints,[],[],parametricVariables);
```

■

Adding advanced constraints is possible also without the dualization procedure, since YALMIP supports the previously discussed image representation. However, for many large scale problems, the model obtained from the dualization is required since it typically is much smaller and numerically robust.

6.2. *Semidefinite relaxations of combinatorial problems*

The original motivation for the whole dualization procedure was the spread of the modelling language to domains where very simple primal form problems needed to be solved, such as semidefinite relaxations of various combinatorial problems.

Example 6.3 We now apply the dualization to the binary quadratic programming problem introduced in Section 3. In this case, we explicitly construct the dualized model, and then solve it.

```
>> X = sdpvar(n,n);
>> P = [X>0, diag(X)==1];p = Q(:)'*X(:);
>> [D,d] = dualize(P,p);
>> solvesdp(D,-d);
```

The price we have to pay for obtaining the dual of the intended primal model, is the computational time required to analyze the model, extract numerical data and form a new symbolic model. The implementation is completely MATLAB and YALMIP-based and aims more for flexibility and generality than raw speed. To illustrate the cost of dualizing a problem, the computation time spent in solving the original inefficient non-dualized model, the time to perform the dualization, and the time spent on solving the dualized problem, is reported in the table below for random Q of growing dimensions. ■

Table 1. Computational results for semidefinite relaxation of the MAXCUT problem in Example 6.3. The table compares the computation time needed to solve the problem using the default interpretation of the model versus applying dualization to solve the problem in the intended primal form. Although the dualization is not for free, it clearly pays off in terms of total computation effort required.

n	`solvesdp(P,p)`	`dualize`	`solvesdp(D,-d)`
30	2.7 sec	0.02 sec.	0.3 sec.
50	6.5 sec	0.02 sec.	0.7 sec.
75	46 sec	0.03 sec.	0.9 sec.
100	$\geq$ 60 sec	0.04 sec.	1.1 sec.
150	$\geq$ 60 sec	0.06 sec.	1.6 sec.
200	$\geq$ 60 sec	0.1 sec.	2.2 sec.
300	$\geq$ 60 sec	0.2 sec.	5.4 sec.
500	$\geq$ 60 sec	0.7 sec.	20 sec.
1000	$\geq$ 60 sec	2.4 sec.	105 sec.

6.3. *D-optimal design experiment design*

Support for logarithmic barrier terms in the objective was motivated by the application in [10].

Example 6.4 The problem which has its basis in a system identification problem for flexible robot models, boils down to a variant of the D-optimal design problem [9].

$$\begin{aligned} \text{minimize} \quad & \log\det\left[\sum_{i=1}^{n}\lambda_i H_i\right]^{-1} \\ \text{subject to} \quad & \lambda \geq 0, \quad \mathbf{1}^T\lambda = 1 \end{aligned} \tag{11}$$

What makes the problem in [10] interesting, in the context of this paper, is the size of the problem. The given matrices $H_i \in \mathbf{S}^d$ are of size 12×12, whereas the decision variable $\lambda \in \mathbf{R}^n$ ranges in size between a couple of hundred up to almost 10^4. A common mistake when approaching (11) is to model the problem as a determinant optimization problem with n variables. This is however a terrible model. Instead, it must be realized that the problem is fairly small, when interpreted in a primal

form. We see this if we manually derive the dual [8].

$$\begin{aligned} \text{minimize} \quad & \log\det W^{-1} - s - d \\ \text{subject to} \quad & W \succeq 0 \\ & W \bullet H_i + s \leq d \end{aligned} \tag{12}$$

This problem has only 79 variables in a dual interpretation (the variables parameterizing the symmetric 12×12 matrix W and the scalar s related to the equality constraint). Instead, there is a large number of linear inequalities. However, the computational complexity of solving an SDP, using a standard SDP solver, is to a large extent driven by the number of dual variables (at least cubic complexity due to the solution of the Newton step), whereas the number of linear inequalities in the dual form only affects complexity linearly (in the construction of the so called Schur matrix, and during line-searches for the linear cone).

By stating the original problem (11) in YALMIP, and applying the automatic dualization, computation time for an instance with $n = 7920$ drops from half an hour for the naive dual interpretation to 15 seconds using [8]. The dualization procedure takes roughly one second.

```
>> lambda = sdpvar(n,1);
>> H=0;for i = 1:n;H = H+Hi{i}*lambda(i);end
>> options =  sdpsettings('dualize',1,'solver','sdpt3');
>> solvesdp([lambda>=0, sum(lambda) == 1], -logdet(H), options)
```

Of course, we know the suitable dual form representation (12) of the problem, but the idea is that there is no need to derive this dual model. Instead, the user can stay in the original application-motivated format, and let the software take care of deriving and solving the suitable form. If additional constraints or variables were to be added to the original problem (11), time-consuming and error-prone manual derivation of a new dual is avoided. ■

7. Conclusions

Automatic translation of conic problems from dual to primal forms can greatly simplify the modelling effort and make a huge impact on the computational effort required to solve the problem. A general algorithm for performing the "dualization" for standard conic (linear, second order cone and semidefinite programming) optimization problems has been implemented in YALMIP, and has already been reported to be successfully employed, and crucial for convenient modelling, in many diverse fields such as control theory, machine learning, system identification and quantum physics.

Acknowledgment

The author would like to acknowledge inspiring discussions and motivation from Nathan Srebro, Michael Tsuk and Erik Wernholt. Significant parts of the work was developed while the author was at the Division of Automatic control, ETH Zürich, Switzerland. Part of the work was performed while the author was visiting the Institute for Mathematical Sciences, National University of Singapore in 2006. The visit was supported by the Institute.

References

[1] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MPS-SIAM series on Optimization. SIAM, Philadelphia, Pennsylvania, 2001.

[2] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM Studies in Applied Mathematics. SIAM, Philadelphia, Pennsylvania, 1994.

[3] M.X. Goemans and D.P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, 42:1115–1145, 1995.

[4] J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004. Available from control.ee.ethz.ch/∼joloef/wiki/pmwiki.php.

[5] J. Löfberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions on Automatic Control*, Accepted for publication, 2008.

[6] P.A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming (Series B)*, 96(2):293–320, 2003.

[7] K.C. Toh. An inexact primal-dual path-following algorithm for convex quadratic SDP. *Mathematical Programming (Series B)*, 112(1):221–254, 2008.

[8] K.C. Toh, R.H. Tütüncü, and M.J. Todd. On the implementation and usage of SDPT3 – a MATLAB software package for semidefinite-quadratic-linear programming, version 4.0, Jul 2006. Available from http://www.math.nus.edu.sg/mattohkc/guide4-0-draft.pdf.

[9] L. Vandenberghe, S. Boyd, and S.P. Wu. Determinant maximization with linear matrix inequality constraints. *SIAM Journal on Matrix Analysis and Applications*, 19(2):499–533, 1998.

[10] E. Wernholt and J. Löfberg. Experiment design for identification of nonlinear gray-box models with application to industrial robots. In *Proceedings of the 46th IEEE Conference on Decision and Control*, New Orleans, Louisiana, December 2007.