2010 IEEE International Symposium on Computer-Aided Control System Design
Part of 2010 IEEE Multi-Conference on Systems and Control
Yokohama, Japan, September 8-10, 2010

Low-rank exploitation in semidefinite programming for control

Rikard Falkeborn, Johan Löfberg and Anders Hansson

Abstract— **Many control related problems can be cast as semidefinite programs but, even though there exist polynomial time algorithms and good publicly available solvers, the time it takes to solve these problems can be long. Something many of these problems have in common, is that some of the variables enter as matrix valued variables. This leads to a low-rank structure in the basis matrices which can be exploited when forming the Newton equations. In this paper, we describe how this can be done, and show how our code can be used when using SDPT3. The idea behind this is old and is implemented in LMI Lab, but we show that when using a modern algorithm, the computational time can be reduced. Finally, we describe how the modeling language YALMIP is changed in such a way that our code can be interfaced using standard YALMIP commands, which greatly simplifies for the user.**

I. INTRODUCTION

Semidefinite programming (SDP) [1] has gained a lot of interest within the control community, and is now a well established mathematical tool in the field. Many fundamental control problems can be cast as semidefinite programs [2], for example proving robust stability [3], [4], [5].

Moreover, since the beginning of the 90s, there exist efficient algorithms to solve these SDPs in a time which is polynomial in the number of variables and constraints [6]. Today, there are several solvers freely available, such as SeDuMi [7], SDPT3 [8] and SDPA [9] just to mention a few.

Although SDPs can be solved in polynomial time, the number of variables in the problems often grow rapidly and the time needed to solve the problems can be substantial, even though the plant size is modest. Because of this fact, tailor-made solvers for various types of problems have been developed, for example for programs derived from the Kalman-Yakubovic-Popov lemma [10], [11], [12], [13], [14].

However, a problem with these tailor-made solvers is that they often are limited to very particular control problems, thus making them non-applicable for more complex problems where only some part of the problem specification happens to have the structure that can be exploited. Additionally, these efficient solvers are typically hard to hook up to user-friendly modeling languages, such as YALMIP [15], which ultimately means that few users actually will benefit from them.

This paper describes the implementation of a structure exploiting assembler for the Schur complement matrix that can be used in for example SDPT3. The assembler utilizes the fact that many problems in systems and control theory have matrix valued variables which lead to low rank structure in the basis matrices. Additionally, we present a new version of YALMIP which will allow the user to describe control problems in a very natural standard YALMIP format, and take care of the intricate communication between SDPT3 and the structure exploiting code.

To be fair, it should be pointed out that the theoretical idea exploited in this paper was incorporated already in LMI Lab [16], which was one of the first public implementations of an SDP solver, tailored specifically for control problems. However, many years of active research in the SDP field has led to much more efficient algorithms, and it is our goal to leverage on this. A major reason for pursuing the idea in this paper is that it is slightly embarrassing for people coming from the SDP field, that the now over 15 year old LMI Lab solver actually outperforms state-of-the-art general purpose SDP solvers on small- to medium-scale control problems.

The notation is standard. We let $A \succ B (A \succeq B)$ denote that $A - B$ is positive (semi)definite, $\mathcal{R}^n$ denotes the set of real vectors of dimension n and $\mathcal{S}^m$ denotes the set of real symmetric matrices of dimension m. The inner product is denoted by $\langle A, B \rangle$ and is for matrices defined as $\operatorname{Tr}\left(A^T B\right)$.

II. SEMIDEFINITE PROGRAMMING

A semidefinite program can be written as

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & F_0 + \sum_{i=1}^{n} F_i x_i = X \\ & X \succeq 0, \end{aligned} \tag{1}$$

where $c, x \in \mathcal{R}^n$ and $X, F_i \in \mathcal{S}^m$.

Almost all modern SDP-solvers today are using interior-point methods. The main parts of these algorithms consist of forming a system of equations to solve for the search directions needed, solve that system of equations and then do a line search in order to find out the appropriate step size. This procedure is then repeated until a stopping criterion is fulfilled. This stopping criterion may for example be that we are sufficiently close to the optimum.

We remark that forming the system of equations can be computationally expensive, and is in some cases by far the most time-consuming part of the algorithm.

If we let the system of equations to be solved in order to get the search directions be

$$H \Delta x = b, \tag{2}$$

where H is the coefficient matrix, which is usually symmetric and Δx is the search direction, the elements are given by $H_{ij} = \langle F_i, U F_j V \rangle$ when using the Helmberg-Kojima-Monteiro (HKM) direction [17], [18], [19], and by $H_{ij} = \langle F_i, W F_j W \rangle$ when using the Nesterov-Todd direction [20].

Here, H_{ij} denotes the ijth element of the matrix H, and U, V and W are scaling matrices.

In SDP programs encountered in systems and control, the majority of variables x_i in (1) do not come from scalar entities but rather as parts of a matrix variable. However, there is no way to inform any of the public solvers about this fact. Instead, we will use an approach where we supply the solver with the code to assemble the Hessian.

To illustrate this, let us take the Lyapunov inequality as an example. The Lyapunov inequality is

$$A^T P + PA + Q \preceq 0, \tag{3}$$

with $Q \succeq 0$. In order to put this inequality on the form (1), which is used by most solvers today, we let $F_0 = -Q$, $F_i = -A^T E_i - E_i A$ where $E_1, \ldots, E_m$ is a basis for $\mathcal{S}^n$, $m = n(n+1)/2$. However, by doing so, we lose a lot of information that can be used in the formulation of the Schur matrix.

The fact that we can choose E_i as any basis for $\mathcal{S}^n$ can be exploited. This means we can choose $E_i = e_k e_l^T + e_l e_k^T$, if the variable x_i is an off-diagonal element and $E_i = e_k e_k^T$ if x_i is an element on the diagonal, where e_i is a unit vector in $\mathcal{R}^n$. Now, let us compute one element of the resulting Schur matrix for the Lyapunov inequality (3), assuming the HKM direction is used. This can be done by

$$\begin{aligned} H_{op} = \langle & A^T\left(e_i e_j^T + e_j e_i^T\right) + \left(e_i e_j^T + e_j e_i^T\right)A, \\ & U\left(A^T\left(e_k e_l^T + e_l e_k^T\right) + \left(e_k e_l^T + e_l e_k^T\right)A\right)V\rangle = \\ & e_k^T AUA^T e_i e_j^T V e_l + e_k^T AUA^T e_j e_i^T V e_l + \\ & e_l^T AUA^T e_i e_j^T V e_k + e_l^T AUA^T e_j e_i^T V e_k + \\ & e_k^T AU e_i e_j^T AV e_l + e_k^T AU e_j e_i^T AV e_l + \\ & e_l^T AU e_i e_j^T AV e_k + e_l^T AU e_j e_i^T AV e_k + \\ & e_i^T AU e_l e_k^T AV e_j + e_j^T AU e_l e_k^T AV e_i + \\ & e_i^T AU e_k e_l^T AV e_j + e_j^T AU e_k e_l^T AV e_i + \\ & e_l^T U e_i e_j^T AVA^T e_j + e_l^T U e_j e_i^T AVA^T e_k + \\ & e_k^T U e_i e_j^T AVA^T e_l + e_k^T U e_j e_i^T AVA^T e_l. \end{aligned} \tag{4}$$

Since $e_i^T B e_j$ is just the ijth element of B, the element H_{op} is just a sum of products of elements from the matrices AUA^T, AU, AVA^T, AV, U and V. Moreover, the matrices involved are the same for all the positions in the Schur matrix. Hence they can be precomputed once in each iteration. This is the structure exploiting idea that was incorporated already in LMI Lab for a projective method. The reason they could exploit it, while modern general purpose solvers fail to, is that the user has to specify the matrices and their position in the constraints in a very detailed, by many regarded cumbersome, fashion.

In this paper we present an extension to the modeling language YALMIP which allows the user to utilize this structure for interior-point methods using the semidefinite programming solver SDPT3 [8].

We remark that this is not limited to symmetric matrix variables and a single Lyapunov inequality, but more general constraints and variables can be used, as will be described in the following section.

III. SDPs considered

In the paper, we consider SDPs where the constraints are on the following form.

$$\begin{aligned} & \sum_{i=1}^{N_{im}} \sum_{j=1}^{N_{jm}} \left(L_{ijm} P_j R_{ijm}^T + R_{ijm} P_j^T L_{ijm}^T\right) + \\ & \qquad \sum_{i=1}^{N_{im}} \sum_{j=1}^{N_{jm}} A_{ijm}^T P_j A_{ijm} + M_{0m} + \\ & \qquad\qquad \sum_{k=1}^{p} M_{km} x_k \preceq 0, \quad m = 1, \ldots, N, \end{aligned} \tag{5}$$

where P_j and x_k are the optimization variables, and where all the matrices are assumed to have suitable dimensions.

We assume the basis matrices for P_j can be written as

$$E_j = \sum_{h=1}^{\alpha_j} \varepsilon_{hj} \delta_{hj}^T, \tag{6}$$

where ε_{hj} and δ_{hj} are assumed to be unit vectors in appropriate vector spaces.

This implies that P_j can be a symmetric matrix, rectangular matrix, matrix with block diagonal structure, tridiagonal structure, skew-symmetric and many more. We assume M_{km} has no exploitable structure.

For easier presentation, we will drop the indeces m, i.e. the indices that indicate which constraint the matrices belong to.

We now show what the elements in the Schur matrix with respect to the j_1th and j_2th elements in P_j are, for the first term in (5). The corresponding element in the Schur matrix is

$$\begin{aligned} & H_{j_1 j_2} = \\ & \left\langle L_{j_1} \sum_{h=1}^{\alpha_{j_1}} \varepsilon_{hj_1} \delta_{hj_1}^T R_{j_1}^T, U L_{j_2} \sum_{h=1}^{\alpha_{j_2}} \varepsilon_{hj_2} \delta_{hj_2}^T R_{j_2}^T V \right\rangle = \\ & \qquad \sum_{h=1}^{\alpha_{j_1}} \sum_{h=1}^{\alpha_{j_2}} \delta_{hj_1}^T R_{j_1}^T U L_{j_2} \varepsilon_{hj_2} \delta_{hj_2}^T R_{j_2}^T V L_{j_1} \varepsilon_{hj_1}. \end{aligned} \tag{7}$$

It is clear that the for the other terms in (5), the expression will be similar. As an example, for the second term in (5), just interchange L_j and R_j, and ε_{hj} and δ_{hj}. We remark that the entry in the Schur matrix for the jth element in P_j, with respect to the first term in (5) and x_k can be written as

$$\begin{aligned} H_{jk} = & \left\langle L_j \sum_{h=1}^{a_j} \varepsilon_{hj} \delta_{hj}^T R_j^T, U M_k V \right\rangle = \\ & \qquad \sum_{h=1}^{a_j} \left\langle \delta_j^T R_j^T U M_k V L_i \varepsilon_j \right\rangle. \end{aligned} \tag{8}$$

Also in this case, the contribution from the other terms in (5) is very similar. It is obvious from the discussion in the previous section what matrices to precompute and that this will speed up the computations. Finally, we remark that for the unstructured matrices, M_k, the entry in the Schur matrix will be

$$H_{k_1k_2} = \langle M_{k_1}, UM_{k_2}V\rangle, \tag{9}$$

just as it is implemented in SDPT3.

As a last remark in this section, we mention that since sparsity in the basis matrices in (9) is exploited by solvers, the more sparsity in the basis matrices, the faster will the computations in (9) be. We also see that the computations in (7) will not be affected by sparsity in the basis matrices. Hence, the more full the basis matrices are, the better it will be to use (7) in order to assemble the Schur matrix. We also mention that a continuous time Lyapunov inequality, where the basis matrices have the form $A^TE_i + E_iA$ will be relatively sparse, will have roughly $4n$ out of n^2 elements that are non-zero, while a discrete time Lyapunov inequality, where the basis matrices are on the form $A^TE_iA - E_i$ will have all n^2 elements full, unless there is some sparsity in A, which indicates that the proposed method will be relatively better for discrete time systems than continuous time systems.

IV. Implementation

The solver SDPT3 allows for the user to provide the solver with a function that handles the assembly of the Schur matrix. We have written a function that computes the Schur matrix as described in Section III. As input, the function takes the matrices R_{ijk}, L_{ijk}, A_{ijk}, M_{0k}, M_{ijk} from (5) and information about the basis matrices in (6). The computations of the elements in the matrix H in (7) are done using mex-files for increased performance. We remark that the case where we compute the element in H where we have two unstructured matrices as in (9), we use the built in function in SDPT3. To specify all these arguments can be cumbersome, but if YALMIP is used, the user does not have to care about specifying any of the low-level input arguments, since this is done automatically by YALMIP, as will be described in the next section.

V. Improvement of the YALMIP language

One of the most important steps is to make the whole framework easily accessible to the casual user. An efficient solver with a cumbersome interface will have little impact in practice. A first step towards incorporation of an efficient structure-exploiting solver for control was the YALMIP interface to the solver KYPD [10]. Although this interface allowed users to describe problems to KYPD in a fairly straightforward fashion, it still required special-purpose commands specific to this solver. The reason for this is that the core idea in YALMIP is that all expressions are evaluated on the fly, and only the basis-matrices and decision variables are kept. In other words, all expressions are immediately disaggregated and knowledge of underlying matrix variables is lost.

To circumvent this, a new version of YALMIP has been developed. To be able to use the efficient solver described in this paper, it is essential that YALMIP keeps track of aggregated matrix variables. Hence, when an expression is evaluated, YALMIP saves information internally about the factors that constitute the constraints, essentially corresponding to the matrices L and R in (5). These factors are also tracked when some basic operations are performed, such as concatenation, addition, subtraction and multiplication. The factors are however not guaranteed to be kept in highly complex manipulations. If we use an operator for which the factor-tracking is not supported, the expression will be disaggregated, and constraints involving this expression will be handled as a standard SDP constraint by the solver.

To summarize, for the user, standard YALMIP code applies and nothing has changed, the only difference is that in some problems structure will automatically be detected and exploited. As an example, the example described in the following section would be coded as

```
P = sdpvar(n);
x = sdpvar(nx,1);
M = M0+x(1)*M1+x(2)*M2+x(3)*M3
F = [A'*P+P*A P*B;B'*P zeros(m)]+M > 0
O = trace(C*P)+c'*x
```

Knowledge about the way the matrix variable P enters the problem will be tracked by YALMIP and exploited.

VI. Computational Results

In this section, we give some computational results that demonstrates that in some cases, it is advantageous to use this way of computing the Schur matrix. The first example is on the following form and is taken from [21]. The SDPs we solve have the following structure

$$\begin{aligned} \min_{x,P} \quad & \langle C, P\rangle + c^Tx \\ \text{s.t.} \quad & \begin{bmatrix} A_i^TP + PA_i & PB_i \\ B_i^TP & 0 \end{bmatrix} + \\ & M_{i,0} + \sum_{k=1}^{n_x} x_kM_{i,k} \succeq 0, \quad i = 1, \dots, n_i, \end{aligned} \tag{10}$$

where the decision variables are $P \in \mathcal{S}^n$ and $x \in \mathcal{R}^{n_x}$. All data matrices were generated randomly, but certain care was taken in order for the optimization problems to be feasible. See [21] for the exact details on how the matrices were generated. This type of LMIs appear in a vast number of analysis problems for linear differential inclusions [2].

The optimization problem (10) can easily be put on the form (5) with

$$L_i = \begin{bmatrix} A_i^T \\ B_i^T \end{bmatrix} \qquad R_i = \begin{bmatrix} I \\ 0 \end{bmatrix}. \tag{11}$$

We solve the problem (10) for increasing numbers of states n. We keep $n_i = 3$ and $n_x = 3$ constant for all the problems. The problem was solved for 10 times for each n and the average solution times are plotted in Figure 1.

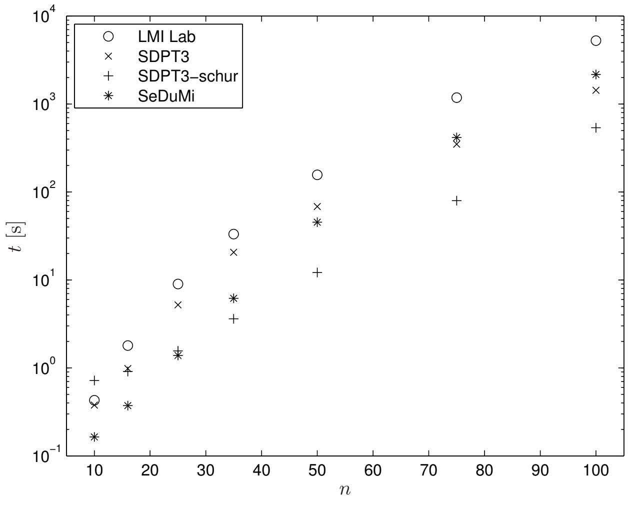

Fig. 1. Averaged computational times.

As can be seen in Figure 1, the solution times decrease if we use the tailor made code for the Schur compilation.

We also give a second, slightly more complicated example. The example is a model reduction algorithm from [22] where semidefinite programming is used to reduce the order of a linear time-invariant (LTI) system. A short description of the algorithm now follows.

It is well known that the H_∞-norm γ of an LTI system can be computed as

$$\min_{\gamma,P} \gamma \tag{12}$$

$$\text{s.t.} \begin{bmatrix} A^TP+PA & PB & C \\ B^TP & -\gamma I & D \\ C^T & D^T & -\gamma I \end{bmatrix} \prec 0. \tag{13}$$

We also know that the difference of two systems $\tilde{G} = G - \hat{G}$ can be written on state space form with the matrices, where $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ are the state-space matrices of a realization of $\tilde{G}$, and analogously with G and $\hat{G}$,

$$\left[\begin{array}{c|c} \tilde{A} & \tilde{B} \\ \hline \tilde{C} & \tilde{D} \end{array}\right] = \left[\begin{array}{cc|c} A & 0 & B \\ 0 & \hat{A} & \hat{B} \\ \hline C & -\hat{C} & D - \hat{D} \end{array}\right]. \tag{14}$$

Now, using $\tilde{G}$ in (12), and by partitioning the matrix P into

$$\begin{bmatrix} P_{11} & P_{12} \\ P_{12}^T & P_{22} \end{bmatrix} \succ 0, \tag{15}$$

and using this in (12), we get,

$$\min_{\hat{A},\hat{B},\hat{C},\hat{D},P_{11},P_{12},P_{22},\gamma} \gamma$$

$$\begin{bmatrix} A^TP+PA & A^TP_{12}+P_{12}\hat{A} & P_{11}B - P_{12}\hat{B} & C^T \\ \hat{A}^TP_{12}^T + P_{12}^TA & \hat{A}^TP_{22}+P_{22}\hat{A} & P_{12}^TB - P_{22}\hat{B} & \hat{C}^T \\ B^TP_{11} - \hat{B}^TP_{12}^T & B^TP_{12} - \hat{B}^TP_{22} & -\gamma I & D^T - \hat{D}^T \\ C & \hat{C} & D - \hat{D} & -\gamma I \end{bmatrix} \prec 0. \tag{16}$$

Since this is a bilinear matrix inequality (BMI), the approach taken in [22] is to fix some matrices to be constant to make it an LMI, solve that optimization problem, and then fix other matrices. This is best described in the following algorithm from [22].

i Start with a $\hat{G}$ obtained from, for example a truncated balanced realization
ii Keeping $(\hat{A}, \hat{B})$ constant, solve (16) subject to (15) with respect to $(P, \hat{C}, \hat{D})$.
iii Keeping (P_{12}, P_{22}) constant, solve (16) subject to (15) with respect to $(P_{11}, \hat{A}, \hat{B}, \hat{C}, \hat{D})$.
iv Repeat *ii* and *iii* until some given convergence criterion is met.

We remark that this procedure does not guarantee global convergence.

Our numerical experience with this algorithm indicates that the numerical properties of the LMIs we need to solve is improved if we let (A, B, C, D) be a balanced realization of G.

We test the algorithm using using SDPT3 both with and without our special purpose Schur-compiler. The systems we test it on are from the Complib library [23]. We remark that for the H_∞-norm to be well defined, the systems must be stable, i.e. all eigenvalues of the A-matrix must have strictly negative real parts. Unfortunately, this is not the case for most of the models in the Complib library. In an attempt to increase the number of models we can use, we shift the spectrum of the A-matrices in some models such that no eigenvalue has larger real part than -1. Results of this is summarized in Table I. In the table, n_x is the number of states in the original model, n_{red} is the number of states in the reduced system, n_u is the number of inputs, n_y is the number of outputs, t_{STRUL} and t_{SDPT3} is the time used by SDPT3 with and without the Schur-compiler. The time is for doing one round of iterations in the algorithm outlined above. The models LAH and JE1 are used without doing any shifting of the spectrum, while the other models where first shifted in order to get stable models. As can be seen in the table, the computational times can be reduced by the use of our code.

TABLE I
COMPARISSON OF TIMES.

Name	n_x	n_{red}	n_u	n_y	t_{STRUL}	t_{SDPT3}	Improvement
LAH	48	18	1	1	209	472	2.26
JE1	24	4	3	5	12.3	26.7	2.17
AC10	48	10	2	2	127	221	1.74
AC13	24	8	3	4	19.8	31.4	1.58
JE2	21	4	3	3	8.81	22.7	2.58
IH	20	10	11	10	22.5	54.2	2.4
CSE1	19	4	2	10	12.3	21.3	1.73

VII. Conclusions

In this paper we have presented a dedicated assembler for the Schur matrix for SDPT3. The Schur matrix is the coefficient matrix that defines the system of equations used in order to solve for the search directions. The assembler utilizes the fact that many semidefinite programs in systems and control theory involve large matrix variables which implies that the basis matrices have a special low rank structure that can be exploited in order reduce the computational burden. We also presented a related extension to the modelling language YALMIP which allows us to keep track of aggregated matrix variables, and exploit these in a solver, something which can be done automatically without any extra input from the user. In two examples, it was demonstrated that using this method can be beneficial. The first example was an academic example where the SDP has the so called KYP structure for increasing sizes of the problem. It this example, the speedup using our code was about five times faster than using SDPT3, SeDuMi and LMI Lab for some sizes of the problem. In the second example, we tested the code on a model reduction algorithm on models from the Complib library. The speed up here was not as good as in the previous example, but still most of the examples are at least twice as fast as SDPT3. There are several contributing factors to this. The major one is that in the first example, we have multiple constraints which include the same variables. This means the time spent on assembling the Schur matrix is three (since we had three constraints) times as large as if we had only had one constraint, but the time to solve for the search directions is only dependent on the number of variables. In the second example we do not have this situation. Finally, we remark that since sparsity in the basis matrices is beneficial for SDPT3, our code would be even better on discrete time problems since these types of problems often have full basis matrices.

REFERENCES

[1] H. Wolkowicz, R. Saigal, and L. Vandenberghe, *Handbook of Semidefinite Programming: Theory, Algorithms, and Applications*. Kluwer Academic Publishers, 2000.

[2] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear matrix inequalities in system and control theory*, ser. Studies in Applied Mathematics. SIAM, 1994, vol. 15.

[3] P. Gahinet, P. Apkarian, and M. Chilali, "Affine parameter-dependent Lyapunov functions and real parametric uncertainty," *IEEE Transactions on Automatic Control*, vol. 41, no. 3, pp. 436 – 442, 1996.

[4] T. Iwasaki and G. Shibata, "LPV system analysis via quadratic separator for uncertain implicit systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 8, pp. 1195 – 1208, 2001.

[5] A. Megretski and A. Rantzer, "System analysis via integral quadratic constraints," *IEEE Transactions on Automatic Control*, vol. 42, no. 6, pp. 819 – 830, 1997.

[6] Y. Nesterov and A. Nemirovsky, "Interior point polynomial methods in convex programming," *Studies in applied mathematics*, vol. 13, 1994.

[7] J. Sturm, "Using SeDuMi 1.02, A Matlab toolbox for optimization over symmetric cones," *Optimization Methods and Software*, vol. 11, no. 1, pp. 625–653, 1999.

[8] R. Tütüncü, K. Toh, and M. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3," *Mathematical Programming*, vol. 95, no. 2, pp. 189–217, 2003.

[9] M. Yamashita, K. Fujisawa, and M. Kojima, "Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0)," *Optimization Methods and Software*, vol. 18, no. 4, pp. 491–505, 2003.

[10] R. Wallin, C.-Y. Kao, and A. Hansson, "A cutting plane method for solving KYP-SDPs," *Automatica*, vol. 44, no. 2, pp. 418 – 429, 2008.

[11] R. Wallin, A. Hansson, and J. H. Johansson, "A structure exploiting preprocessor for semidefinite programs derived from the Kalman-Yakubovich-Popov lemma," *IEEE Transactions on Automatic Control*, vol. 54, no. 4, pp. 697–704, April 2009.

[12] C.-Y. Kao, A. Megretski, and U. Jönsson, "Specialized fast algorithms for IQC feasibility and optimization problems," *Automatica*, vol. 40, no. 2, pp. 239 – 252, 2004.

[13] Z. Liu and L. Vandenberghe, "Low-rank structure in semidefinite programs derived from the KYP lemma," in *Proceedings of the 46th IEEE Conference on Decision and Control*. Citeseer, 2007.

[14] L. Vandenberghe, V. Balakrishnan, R. Wallin, A. Hansson, and T. Roh, "Interior-point algorithms for semidefinite programming problems derived from the KYP lemma," *Positive Polynomials in Control, Lectures Notes in Control and Information Science. Springer*, 2004.

[15] J. Löfberg, "YALMIP: a toolbox for modeling and optimization in MATLAB," *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pp. 284–289, 2004.

[16] P. Gahinet, A. Nemirovski, A. Laub, and M. Chilali, "LMI control toolbox," *The MathWorks Inc*, 1995.

[17] C. Helmberg, F. Rendl, R. Vanderbei, and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization*, vol. 6, pp. 342–361, 1996.

[18] M. Kojima, S. Shindoh, and S. Hara, "Interior-point methods for the monotone semidefinite linear complementarity problem in symmetric matrices," *SIAM Journal on Optimization*, vol. 7, p. 86, 1997.

[19] R. Monteiro, "Primal–dual path-following algorithms for semidefinite programming," *SIAM Journal on Optimization*, vol. 7, no. 3, pp. 663–678, 1997.

[20] Y. E. Nesterov and M. J. Todd, "Self-scaled barriers and interior-point methods for convex programming," *Mathematics of Operations Research*, vol. 22, no. 1, pp. 1–42, 1997.

[21] J. H. Johansson and A. Hansson, "An inexact interior-point method for system analysis," *International Journal of Control*, To appear.

[22] A. Helmersson, "Model reduction using LMIs," in *Proceedings of the 33rd IEEE Conference on Decision and Control*, Orlando, Florida, Feb. 1994, pp. 3217–3222.

[23] F. Leibfritz, "COMPLeIB, COnstraint Matrix-optimization Problem LIbrary-a collection of test examples for nonlinear semidefinite programs, control system design and related problems," Technical report, Universität Trier, 2003, Tech. Rep., 2003.