Low-rank exploitation in semidefinite programming for control

Rikard Falkeborn, Johan Löfberg and Anders Hansson

RESEARCH ARTICLE

Low-rank exploitation in semidefinite programming for control

Rikard Falkeborn, Johan Löfberg and Anders Hansson
Division of Automatic Control, Department of Electrical Engineering
Linköping University, SE-581 83 Sweden,
{falkeborn,johanl,hansson}@isy.liu.se

(*Received 00 Month 200x; final version received 00 Month 200x*)

Many control related problems can be cast as semidefinite programs. Even though there exist polynomial time algorithms and excellent publicly available solvers, the time it takes to solve these problems can be excessive. What many of these problems have in common, in particular in control, 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, called STRUL, can be used in conjunction with the semidefinite programming solver SDPT3. The idea behind the structure exploitation is classical and is implemented in LMI Lab, but we show that when using a modern semidefinite programming framework such as SDPT3, the computational time can be significantly reduced. Finally, we describe how the modeling language YALMIP has been changed in such a way that our code, which can be freely downloaded, can be interfaced using standard YALMIP commands. This greatly simplifies modeling and usage.

1 Introduction

Semidefinite programming (SDP) (Wolkowicz et al. 2000) has gained significant 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 (Boyd et al. 1994), with robust stability analysis being one of the most common application (Gahinet et al. 1996, Iwasaki and Shibata 2001, Megretski and Rantzer 1997, Ben-Tal and Nemirovski 2001).

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 (Nesterov and Nemirovsky 1994). Today, there are several solvers freely available, such as SeDuMi (Sturm 1999), SDPT3 (Tütüncü et al. 2003) and SDPA (Yamashita et al. 2003).

Although SDPs can be solved in polynomial time, the number of variables in the problems are often huge 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 (Wallin et al. 2008, 2009, Kao et al. 2004, Liu and Vandenberghe 2007, Vandenberghe et al. 2004). 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 solvers are typically hard to interface with user-friendly modeling languages, such as YALMIP (Löfberg 2004). It ultimately means that few users actually will benefit from them.

This paper describes the implementation STRUL (STRUctured LMIs), a structure exploiting assembler for the Schur 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 in STRUL.

To be fair, it should be pointed out that the theoretical idea exploited in this paper was incorporated already in LMI Lab (Gahinet et al. 1995), 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 some small- to medium-scale control problems. It is our goal to eliminate this performance discrepancy, while at the same time leverage on the intuitive modelling capabilities of YALMIP.

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

2 Semidefinite programming

A semidefinite program can be written as[1]

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

where $c, x \in \mathbb{R}^n$ and $F_i \in \mathbb{S}^m$. The constraint (1) in the SDP is called a linear matrix inequality (LMI).

Almost all modern SDP-solvers today use 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 the decrease in objective value is sufficiently small.

Forming the system of equations can be computationally expensive, and is in some cases by far the most time-consuming part of the algorithm. Let the system of equations to be solved in order to find the search directions be

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

where H is the so called Schur matrix (usually symmetric) and Δx is the search direction. The elements are given by

$$H_{ij} = \langle F_i, UF_jV\rangle \tag{3}$$

when using the Helmberg-Kojima-Monteiro (HKM) direction (Helmberg et al. 1996, Kojima et al. 1997, Monteiro 1997), and by

$$H_{ij} = \langle F_i, WF_jW\rangle \tag{4}$$

[1]This form is most often used in the control community. It should be mentioned that it is a slight variation of what normally is called the dual form in the semidefinite programming community.

when using the Nesterov-Todd direction (Nesterov and Todd 1997). Here, H_{ij} denotes the ijth element of the matrix H, and U, V and W are scaling matrices.

In LMIs 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, as described in detail below. However, there is no way to inform any of the public solvers about this fact, so that they can exploit this when forming (4). Instead, we will use an approach where we supply the solver with the code to assemble the Schur matrix.

To illustrate the basic idea, let us consider a Lyapunov inequality as an example. The Lyapunov inequality is

$$A^T P + PA + Q \preceq 0, \tag{5}$$

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

We can exploit the fact that we can choose E_i as any basis for $\mathbb{S}^n$. This means we can choose $E_i = e_k e_l^T + e_l e_k^T$, if the variable x_i corresponds to the off-diagonal element at position (k, l) and (l, k) and $E_i = e_k e_k^T$ if x_i is the kth diagonal element, where e_i is a unit vector in $\mathbb{R}^n$. Now, let us compute one element of the resulting Schur matrix for the Lyapunov inequality (5), assuming the HKM direction is used. If $E_r = e_k e_l^T + e_l e_k^T$ and $E_p = e_i e_j^T + e_j e_i^T$, the element in the Schur matrix corresponding to the variables x_r and x_p evaluates to

$$\begin{aligned}
H_{rp} = \big\langle & A^T \left(e_i e_j^T + e_j e_i^T\right) + \left(e_i e_j^T + e_j e_i^T\right) A, \\
& U \left(A^T \left(e_k e_l^T + e_l e_k^T\right) + \left(e_k e_l^T + e_l e_k^T\right) A\right) V \big\rangle = \\
& e_k^T AUA^T e_i e_j^T V e_l + e_k^T AUA^T e_j e_i^T V e_l + \\
& e_l^T AUA^T e_i e_j^T V e_k + e_l^T AUA^T e_j e_i^T V e_k + \\
& e_k^T AU e_i e_j^T AV e_l + e_k^T AU e_j e_i^T AV e_l + \\
& e_l^T AU e_i e_j^T AV e_k + e_l^T AU e_j e_i^T AV e_k + \\
& e_i^T AU e_l e_k^T AV e_j + e_j^T AU e_l e_k^T AV e_i + \\
& e_i^T AU e_k e_l^T AV e_j + e_j^T AU e_k e_l^T AV e_i + \\
& e_l^T U e_i e_j^T AVA^T e_j + e_l^T U e_j e_i^T AVA^T e_k + \\
& e_k^T U e_i e_j^T AVA^T e_l + e_k^T U e_j e_i^T AVA^T e_l.
\end{aligned} \tag{6}$$

Since $e_i^T B e_j$ is just the ijth element of B, the element H_{rp} 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 users to utilize this structure for interior-point methods using the semidefinite programming solver SDPT3 (Tütüncü et al. 2003). Initial results were reported in Falkeborn et al. (2010).

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.

3 SDPs considered

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

$$\sum_{i=1}^{N_{im}}\sum_{j=1}^{N_{jm}}\left(L_{ijm}P_jR_{ijm}^T+R_{ijm}P_j^TL_{ijm}^T\right)+ \sum_{i=1}^{N_{im}}\sum_{j=1}^{N_{jm}}A_{ijm}^TP_jA_{ijm}+M_{0m}+ \sum_{k=1}^{p}M_{km}x_k\preceq 0,\quad m=1,\dots,N, \tag{7}$$

where P_j and x_k are the optimization variables, and all the matrices are assumed to have suitable dimensions. This structure arise in a large number of stability analysis and synthesis problems in control and systems theory.

We assume the basis matrices for P_j can be written as

$$E_j=\sum_{h=1}^{\alpha_j}\varepsilon_{hj}\delta_{hj}^T, \tag{8}$$

where ε_{hj} and δ_{hj} are assumed to be unit vectors (columns of the identity matrix) 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 indices m, i.e. the indices that indicate which constraint the matrices belong to, and the indices i.

We now show what the elements in the Schur matrix with respect to the elements in P_j corresponding to the basis matrices E_{j_1} and E_{j_2} in (8), for the first term in (7). The corresponding element in the Schur matrix is

$$H_{j_1j_2}=\left\langle L_j\sum_{h=1}^{\alpha_{j_1}}\varepsilon_{hj_1}\delta_{hj_1}^TR_j^T,UL_j\sum_{h=1}^{\alpha_{j_2}}\varepsilon_{hj_2}\delta_{hj_2}^TR_j^TV\right\rangle=\sum_{h=1}^{\alpha_{j_1}}\sum_{h=1}^{\alpha_{j_2}}\delta_{hj_1}^TR_j^TUL_j\varepsilon_{hj_2}\delta_{hj_2}^TR_j^TVL_j\varepsilon_{hj_1}. \tag{9}$$

It is clear that for the other terms in (7), the expression will be similar. As an example, for the second term in (7), 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 (7) and x_k with respect to the last term in (7) can be written as

$$H_{jk}=\left\langle L_j\sum_{h=1}^{a_j}\varepsilon_{hj}\delta_{hj}^TR_j^T,UM_kV\right\rangle=\sum_{h=1}^{a_j}\left\langle\delta_{hj}^TR_j^TUM_kVL_j\varepsilon_{hj}\right\rangle. \tag{10}$$

Also in this case, the contribution from the other terms in (7) is very similar. In this case, $R_j^TUM_kVL_j$ is the same for all the elements in H with respect to P_j and x_k. Finally, we remark

that for the unstructured matrices, M_k, the entry in the Schur matrix will be

$$H_{k_1 k_2} = \langle M_{k_1}, U M_{k_2} V \rangle \,, \tag{11}$$

just as it is implemented in e.g. SDPT3.

As a last remark in this section, we mention that since sparsity in the basis matrices M_{k_1} and M_{k_2} in (11) is exploited by solvers, the more sparsity in the basis matrices, the faster will the computations in (11) be, using a standard compilation scheme. The computations in (9) however 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 (9) in order to assemble the Schur matrix. We also mention that a continuous time Lyapunov inequality, where the basis matrices have the form $A^T E_i + E_i A$ will be relatively sparse and have roughly $4n$ non-zero elements out of n^2, while a discrete time Lyapunov inequality, where the basis matrices are on the form $A^T E_i A - E_i$ will have all n^2 elements full, unless there is some sparsity in A. This indicates that the proposed method will be relatively better for discrete time systems than continuous time systems, since a sparsity exploiting solver will have nothing to exploit.

4 Comparison with other assembling schemes

Consider again the Lyapunov problem (5). We will now compare the standard method with using the lowrank properties, another lowrank technique which is already implemented in SDPT3, and similar to what is used in DSDP (Benson and Ye 2005) (note that DSDP primarily is intended for problems with a sparse dual, which typically not is the case in our targeted applications).

First let us estimate the number of flops that is needed to compute the Schur matrix using the technique described in the previous section.

- All matrix products AUA^T, AVA^T, AU and AV can be computed once in each iteration at a cost of $\mathcal{O}(n^3)$ flops.
- Each element in H in (6) can be computed in 31 flops.
- Since there are roughly $n^4/8$ unique entries in H, the total cost is approximately $31 * n^4/8 \approx 4n^4$ flops in total for assembling H.

The unstructured way of doing this, as in (3), cost $\mathcal{O}(n^6)$ operations (each of the $\mathcal{O}(n^4)$ elements in H requires an inner product between two matrices of dimension n). However, we remark that this estimation assumes the basis matrices to be full. If the basis matrices are sparse, which is often the case in applications, then the time needed to compute the Schur matrix can be lowered substantially (Fujisawa et al. 1997).

The third way to compute the entries of the Schur matrix is to use the fact that the basis matrices can be factored as $F_i = V_i D_i V_i^T$ with $V_i \in \mathbb{R}^{n \times r}$ and $D_i \in \mathbb{R}^{r \times r}$ where r is the rank of F_i. This way, each element of the Schur matrix can be formed using $\operatorname{Tr} V_j^T V D_i V_i^T U V_j D_j$, something that can be done in $\mathcal{O}(n^4)$ if the rank of F_i and F_j is small. Note that factorization of the matrices is left out from this analysis.

5 Implementation

The solver SDPT3 allows for the user to provide the solver with a function that handles the assembly of the Schur matrix, or parts of it. To leverage on this, we have developed a software module called STRUL, which computes the Schur matrix as described in Section 3. As input, the function takes the matrices R_{ijk}, L_{ijk}, A_{ijk}, M_{0k}, M_{ijk} from (7) and information about the basis matrices in (8). The computations of the elements in the matrix H in (9) are done using mex-files to increase performance. Entries related to unstructured data is left to SDPT3 to

handle. To manipulate all this data and specify arguments in the solver would be cumbersome and impractical to do manually. Instead, the modeling language YALMIP has been extended to keep track of structured variables, and communicate this information to the solver, and our Schur assembler, automatically.

6 Extension of the YALMIP modeling framework

One of the most important contribution of the presented work is to make the whole framework easily accessible to casual users. An efficient solver with a cumbersome interface has 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 (Wallin et al. 2009). 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. Additionally, the KYPD solver is only applicable to very restricted LMI problems arising in control theory.

While new solvers typically are easy to add to the list of supported solvers in YALMIP, it is a somewhat more involved project to add support for solvers which exploit structure arising from matrix variables. 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 scalar 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 method 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 (7). These factors are also tracked when some basic operations are performed, such as concatenation, addition, subtraction, multiplication and lifting from complex-valued LMIs to real-valued LMIs. The factors are 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 unstructured SDP constraint by the underlying solver, in our case SDPT3.

To summarize, for the user, standard YALMIP code applies and nothing has changed, the only difference is that in some problems its structure is automatically detected and exploited. As an example, the extended balanced truncation example described in (13) would be coded as[1]

```
P = blkdiag(sdpvar(n/3),sdpvar(n/3),sdpvar(n/3));
S = sdpvar(n,n,'full')
Constraints = [P A*S B;
               S'*A' S+S'-P zeros(n,m);
               B' zeros(m,n) eye(m)]>0
Objective = trace(P)
options = sdpsettings('solver','strul')
solvesdp(Constraints,Objective,options)
```

Knowledge about the way the matrix variables P and R enters the problem will be tracked by YALMIP and exploited. For reference, the following code implements the same algorithm using LMI Lab.

```
setlmis([]);
P = lmivar(1,[n/3 1;n/3 1;n/3 1]);
S = lmivar(2,[n n]);
lmiterm([-1 1 1 P],1,1);
```

[1]For a complete on-line manual, please see `http://users.isy.liu.se/johanl/yalmip/`

```
lmiterm([-1 1 2 S],A,1);
lmiterm([-1 1 3 0],B);
lmiterm([-1 2 2 S],1,1,'s');
lmiterm([-1 2 2 P],-1,1);
lmiterm([-1 3 3 0],eye(m));
lmisys = getlmis
c = mat2dec(lmisys,eye(n),zeros(n))
[copt,xopt] = mincx(lmisys,c);
```

7 Computational Results

In this section, we give some computational results that illustrate that in many cases in control theory, it is advantageous to use the proposed way of computing the Schur matrix in combination with a modern SDP framework. All computations were performed on a 3GHz desktop PC with 2Gb memory, using MATLAB 7.9 and SDPT3 4.

The first example is taken from (Johansson and Hansson 2010). The SDPs we solve have the following structure

$$\begin{aligned} \min_{x,P} \quad & \langle C, P\rangle + c^T x \\ \text{s.t.} \quad & \begin{bmatrix} A_i^T P + PA_i & PB_i \\ B_i^T P & 0 \end{bmatrix} + M_{i,0} + \sum_{k=1}^{n_x} x_k M_{i,k} \succeq 0, \quad i = 1, \ldots, n_i, \end{aligned} \tag{12}$$

where the decision variables are $P \in \mathbb{S}^n$ and $x \in \mathbb{R}^{n_x}$. All data matrices were generated randomly, but certain care was taken in order for the optimization problems to be feasible. See (Johansson and Hansson 2010) for exact details on how the matrices were generated. This type of LMIs appear in a vast number of analysis problems for linear differential inclusions (Boyd et al. 1994).

The optimization problem (12) can easily be put on the form (7) with $L_i = \begin{bmatrix} A_i^T \\ B_i^T \end{bmatrix}$ and $R_i = \begin{bmatrix} I \\ 0 \end{bmatrix}$. These are the factors that are automatically extracted and used by YALMIP when it communicates with SDPT3 and STRUL. We solve the problem (12) for increasing numbers of states n. We keep $n_i = 3$ and $n_x = 3$ constant for all the problems. The problem was solved 10 times for each n and the average solution times are reported in Figure 1. As can be seen in Figure 1, the solution times decrease significantly if we use the tailor made code for the Schur compilation.

Our second example implements the balanced truncation algorithm for discrete-time systems in (Sandberg 2010). In the paper, the following SDP is introduced

$$\begin{aligned} \min_{S,P} \quad & \operatorname{Tr} P \\ \text{s.t.} \quad & \begin{bmatrix} P & AS & B \\ S^T A^T & S + S^T - P & 0 \\ B^T & 0 & I \end{bmatrix} \succeq 0, \end{aligned} \tag{13}$$

Compared to standard LMIs arising in balanced truncation, the constraint in the SDP above introduces a new set of variables $S \in \mathbb{R}^{n\times n}$, and it is argued that this extra degree of freedom can be beneficial if the extended Gramian $P \in \mathbb{S}^n$ is forced to have structure for some reason. To test our solver, we create random problem instances with growing dimensions of $A \in \mathbb{R}^{n\times n}$ with $B \in \mathbb{R}^{n\times 1}$. The extended Gramian P is defined as a block-diagonal matrix with three blocks P_1, P_2 and P_3 of equal size (hence we only generate instances of size multiples of 3). The structure

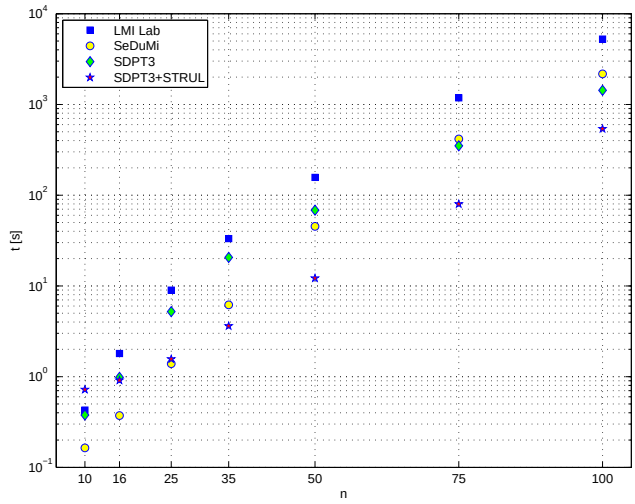

Figure 1. Averaged computational times for the random KYP example. For medium- to large-scale problems, the proposed solver is roughly an order of magnitude faster than LMI Lab, and a couple of factors faster than the general-purpose state-of-the-art SDP solvers SDPT3 and SeDuMi.

our solver and YALMIP has to track and exploit is thus the block-diagonal concatenation of the matrix variables P_i, and the full matrix S.

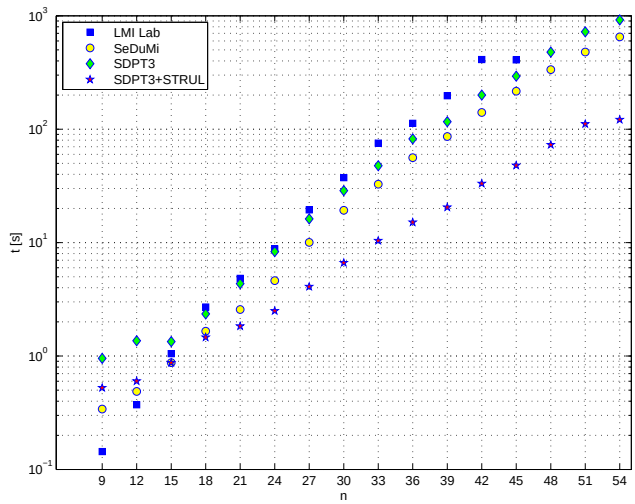

Figure 2. Averaged computational times for the extended balanced truncation example. For increasing problem sizes, the proposed solver is signficantly faster than both LMI Lab and the general-purpose state-of-the-art SDP solvers SDPT3 and SeDuMi. For $n > 45$, LMI Lab failed to compute solutions (out of memory).

Finally, we study a slightly more involved scenario. The example is based on a model reduction algorithm by (Helmersson 1994) 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 a continuous-time LTI system can be computed as

$$\begin{aligned} &\min_{\gamma,P} \gamma \\ &\text{s.t.} \begin{bmatrix} A^TP + PA & PB & C \\ B^TP & -\gamma I & D \\ C^T & D^T & -\gamma I \end{bmatrix} \preceq 0, \;\; P \succeq 0. \end{aligned} \tag{14}$$

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 $\hat{G}$,

$$\tilde{G} = \left[\begin{array}{c|c} \tilde{A} & \tilde{B} \\ \hline \tilde{C} & \tilde{D} \end{array}\right] = \left[\begin{array}{cc|c} A & 0 & B \\ 0 & \hat{A} & \hat{B} \\ \hline C & -\hat{C} & D-\hat{D} \end{array}\right], G = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right], \hat{G} = \left[\begin{array}{c|c} \hat{A} & \hat{B} \\ \hline \hat{C} & \hat{D} \end{array}\right]. \tag{15}$$

Now, using $\tilde{G}$ in (14), and by partitioning the matrix P into

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{12}^T & P_{22} \end{bmatrix} \succeq 0, \tag{16}$$

and using this in (14), we obtain an optimization problem to minimize the H_∞-norm of the model error $G - \hat{G}$.

$$\begin{aligned} &\min_{\hat{A},\hat{B},\hat{C},\hat{D},P,\gamma} \gamma \\ &\text{s.t.} \begin{bmatrix} A^TP + PA & A^TP_{12} + P_{12}\hat{A} & P_{11}B - P_{12}\hat{B} & C^T \\ \hat{A}^TP_{12}^T + P_{12}^TA & \hat{A}^TP_{22} + P_{22}\hat{A} & P_{12}^TB - P_{22}\hat{B} & \hat{C}^T \\ B^TP_{11} - \hat{B}^TP_{12}^T & B^TP_{12} - \hat{B}^TP_{22} & -\gamma I & D^T - \hat{D}^T \\ C & \hat{C} & D - \hat{D} & -\gamma I \end{bmatrix} \preceq 0, \ P \succeq 0. \end{aligned} \tag{17}$$

Since this is a bilinear matrix inequality (BMI), the approach taken in (Helmersson 1994) 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 (Helmersson 1994).

i Start with a $\hat{G}$ obtained from, for example a truncated balanced realization
ii Keeping $(\hat{A}, \hat{B})$ constant, solve (17) subject to (16) with respect to $(P, \hat{C}, \hat{D})$.
iii Keeping (P_{12}, P_{22}) constant, solve (17) subject to (16) 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.

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

We test the algorithm using SDPT3 both with and without our Schur assembler. The systems we test it on are from the COMPleib library (Leibfritz 2004). 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 COMPleib library. In an attempt to increase the number of models we can experiment with, we shift the spectrum of the A-matrices in some models such that no eigenvalue has larger real part than -1. Results from the tests are summarized in Table 1. In the table, n_x is the number of states in the original model $A \in \mathbb{R}^{n_x \times n_x}$, n_{red} is the number of states in the reduced system $A \in \mathbb{R}^{n_{red} \times n_{red}}$, n_u is the number of inputs $B \in \mathbb{R}^{n_x \times n_u}$, n_y is the number of outputs $C \in \mathbb{R}^{n_y \times n_x}$, $t_{\text{SDPT3+STRUL}}$ and t_{SDPT3} is the time (averaged per iteration in the algorithm above) used by SDPT3 with and without the Schur assembler. The models LAH and JE1 are used without any shifting of the spectrum, while the other models where first shifted in order to obtain stable models. As can be seen in the table, the computational times can be reduced by the use of our code, also on these relatively small problems.

Table 1. Comparison of solution times in the model reduction example. The time reported is the average time to solve the problems (*ii*) and (*iii*) in each iteration of the algorithm. By using the proposed Schur assembler, the computation time can be reduced by a factor of 2 on most problems.

Name	n_x	n_{red}	n_u	n_y	$t_{\text{SDPT3+STRUL}}$	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

8 Conclusions

A dedicated Schur assembler used in conjunction with SDPT3 has been developed. The Schur matrix is the coefficient matrix that defines the system of equations for the search directions. The assembler utilizes the fact that many semidefinite programs in systems and control theory involve large matrix variables. This implies that the basis matrices have a special low rank structure that is exploited in order to reduce the computational burden. We also presented a related extension to the modeling language YALMIP which allows us to keep track of aggregated matrix variables, and exploit these in our solver, something which can be done automatically without any extra input from the user. In three examples, it was demonstrated that the proposed framework can be beneficial. The first example was an academic example where the SDP has a so called KYP structure. In this example, the speedup using our code was about five times compared to SDPT3, SeDuMi and LMI Lab for large-scale problems. In the second example, we illustrated how the proposed approach was up to an order of magnitude faster than the other compared solvers, and without problems managed to solve problems where LMI Lab suffered from memory problems. In the final example, we tested the code on a model reduction algorithm on models from the COMPleib library. The speed up here was not as striking as in the previous examples, but still more than half of the problems are solved at least twice as fast compared to standard SDPT3.

Finally, we remark that since sparsity in the basis matrices is exploited efficiently in SDPT3, our code would be even more beneficial on discrete time problems since these types of problems often have full basis matrices.

References

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.

S.J. Benson and Y. Ye. DSDP5: Software for semidefinite programming. *ACM Trans. Math. Software*, 2005.

S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, 1994.

R. Falkeborn, J. Löfberg, and A. Hansson. Lowrank exploitation in semidefinite programming for control. In *Proceedings of the 15th IEEE International Symposium on Computer Aided Control System Design*, Yokohama, Japan, 2010.

K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming*, 79(1):235–253, 1997.

P. Gahinet, A. Nemirovski, AJ Laub, and M. Chilali. LMI control toolbox. *The MathWorks Inc*, 1995.

P. Gahinet, P. Apkarian, and M. Chilali. Affine parameter-dependent Lyapunov functions and

real parametric uncertainty. *IEEE Transactions on Automatic Control*, 41(3):436 – 442, 1996.

C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6:342–361, 1996.

A. Helmersson. Model reduction using LMIs. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 3217–3222, Orlando, Florida, February 1994.

T. Iwasaki and G. Shibata. LPV system analysis via quadratic separator for uncertain implicit systems. *IEEE Transactions on Automatic Control*, 46(8):1195 – 1208, August 2001.

J. H. Johansson and A. Hansson. An inexact interior-point method for system analysis. *International Journal of Control*, 83(3):601–616, March 2010.

C.-Y. Kao, A. Megretski, and U. Jönsson. Specialized fast algorithms for IQC feasibility and optimization problems. *Automatica*, 40(2):239 – 252, February 2004.

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

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, Technical report, Universität Trier, 2004.

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*, 2007.

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

A. Megretski and A. Rantzer. System analysis via integral quadratic constraints. *IEEE Transactions on Automatic Control*, 42(6):819 – 830, June 1997.

R.D.C. Monteiro. Primal–dual path-following algorithms for semidefinite programming. *SIAM Journal on Optimization*, 7(3):663–678, 1997.

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

Y. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

H. Sandberg. An Extension to Balanced Truncation With Application to Structured Model Reduction. *IEEE Transactions on Automatic Control*, 55(4):1038–1043, April 2010. .

J.F. Sturm. Using SeDuMi 1.02, A Matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1):625–653, 1999.

R.H. Tütüncü, K.C. Toh, and M.J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming*, 95(2):189–217, 2003.

L. Vandenberghe, V.R. 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.

R. Wallin, C.-Y. Kao, and A. Hansson. A cutting plane method for solving KYP-SDPs. *Automatica*, 44(2):418 – 429, February 2008.

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*, 54(4):697–704, April 2009.

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

M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). *Optimization Methods and Software*, 18(4):491–505, 2003.