On the application of different numerical methods to obtain null-spaces of polynomial matrices. Part 2: block displacement structure algorithms.

J.C. Zúñiga and D. Henrion

***Abstract*— Motivated by some control problems requiring the extraction of the null-space of large polynomial matrices, we introduce two *fast* numerical linear algebra algorithms to perform this task. These fast algorithms are based on the displacement structure theory and the generalized Schur algorithm. We present a full analysis of the performance of both algorithms and a comparison with the performance of the classical algorithms proposed in the first part of this work [12]. We also discuss briefly the numerical stability of the algorithms.**

I. INTRODUCTION

The computation of null-spaces of large polynomial matrices arises when solving several control problems. As an example, consider the connected mass-spring system in figure 1 where p is the number of masses, u is the force applied to the first mass and m_i, k_i and x_i for $i = 1, 2, \ldots, p$ are the values of the masses, the constants of the springs and the mass positions respectively.

The dynamics of this system are governed by the second-order differential equation

$$M \frac{\mathrm{d}^2}{\mathrm{d}t^2} x + Kx = Bu, \qquad (1)$$

where $x = [x_1 \ x_2 \ \ldots \ x_p]^T$, $B = [1 \ 0 \ \ldots \ 0]^T$, $M = \mathrm{diag}\{m_1, m_2, \ldots, m_p\}$ and

$$K = \begin{bmatrix} k_1 & -k_1 & & & \\ -k_1 & k_1 + k_2 & & & \\ & & \ddots & & \\ & & & k_{p-2} + k_{p-1} & -k_{p-1} \\ & & & -k_{p-1} & k_{p-1} + k_p \end{bmatrix}.$$

Let the output y be x_p, the position of the last mass, so that the output equation is given by $y = Cx$ where $C = [0 \ 0 \ \cdots \ 1]$. Suppose we want to obtain the transfer function from u to y. This transfer function can be obtained from the null-space of a polynomial matrix. The Laplace transform of equation (1) is given by $D(s)x = Bu$, where $D(s) = Ms^2 + K$. So, the system transfer function is given by

$$G(s) = CD^{-1}(s)B = C\frac{N(s)}{d(s)} = \frac{n(s)}{d(s)},$$

J.C. Zúñiga and D. Henrion are with LAAS-CNRS, 7 Avenue du Colonel Roche, 31077 Toulouse, France.

D. Henrion is also with the Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 4, 18208 Prague, Czech Republic, and with the Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 16627 Prague, Czech Republic.

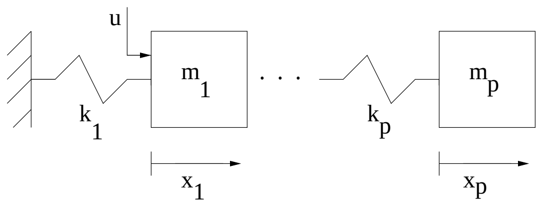

Fig. 1. Connected mass-spring system of p masses.

from which it follows that

$$D^{-1}(s)B = \frac{N(s)}{d(s)}, \quad D(s)N(s) = Bd(s).$$

So, from the vector of minimal degree of the null-space of $[D(s) \ -B]$, we obtain the numerator $n(s)$ and the denominator $d(s)$ of the transfer function as follows:

$$\begin{bmatrix} D(s) & -B \end{bmatrix} \begin{bmatrix} \times \\ \times \\ \vdots \\ n(s) \\ d(s) \end{bmatrix} = 0. \qquad (2)$$

Other physical systems can be modeled as the mass-spring system described above. Consider for example an axially vibrating nonconservative rod with p nodes [1] or a building with p floors [8]. For these kind of systems p is usually large.

In the first part of this work [12] we analyzed two different block Toeplitz algorithms to obtain the null-space of a polynomial matrix. One of these algorithms is based on the LQ factorization (the dual of the QR factorization [2]) and the other on the Column Echelon Form (CEF) of successive block Toeplitz matrices of increasing size. We have seen that the number of columns η of the Toeplitz matrices depends on the dimension and the degree of the polynomial matrix. We have also seen that the complexity of both algorithms is in $O(\eta^3)$. In some control problems, for example the system described above, η could be large and then the previous algorithms may be inefficient.

In order to reduce the algorithmic complexity, in this paper we re-design the Toeplitz algorithms `LQ` and `CEF` in [12] by applying the displacement structure theory. A method based on the generalized Schur algorithm is used to obtain the LQ or the CEF of the block Toeplitz matrices at each step. The new algorithms are called fast algorithms because their complexity can be reduced asymptotically to $O(\eta^2)$. For a comprehensive review of displacement structure and fast algorithms see [4] and [5]. Other references for applications of block displacement structure theory are [6] and [7]. See also the working notes [9].

In section 2 we present some basics and a block version of the generalized Schur algorithm. In section 3 we apply the generalized Schur algorithm to obtain a fast LQ factorization and a fast CEF method. Then in section 4 we develop our algorithms to obtain null-spaces. In section 5 we use as a numerical example the mass-spring system of Figure 1 to compare the performance of the algorithms.

II. Preliminaries

The displacement structure theory is covered in detail in [4], [5], see also [6]. Here we present some results for the sake of clarity.

We define the block displacement of an $m \times n$ matrix A as

$$\nabla A = A - F_1 A F_2^T$$

where the strictly lower block triangular matrices F_1 and F_2 of dimensions $m \times m$ and $n \times n$ respectively are called the block displacement operators.

The pair (X, Y) consisting of the constant $m \times \alpha$ matrix X and the constant $n \times \alpha$ matrix Y such that $\nabla A = XY^T$ is called a generator of A. Integer α is the length of the generator. The minimal length $\alpha_{\min}$ of the generator is called the displacement rank, i.e. $\operatorname{rank} \nabla A = \alpha_{\min}$.

In the particular case of a symmetric $n \times n$ matrix A, we can define a symmetric block displacement

$$\nabla A = A - FAF^T = X \Sigma X^T \tag{3}$$

with

$$\Sigma = I_p \oplus -I_q = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}$$

where p and q are suitable dimensions. We call X the symmetric generator of A.

The generator (X, Y) of A is not unique. Any nonsingular $\alpha \times \alpha$ matrix M yields another generator (XM, YM^{-T}) of A. For symmetric generators, M should be Σ-unitary, namely, $M^T \Sigma M = M \Sigma M^T = \Sigma$. In particular, we can find a transformation matrix M such that the new generator becomes proper.

Definition 1: If a generator (X, Y) has the form

$$X = \begin{bmatrix} X_{11} & 0 \\ X_{21} & X_{22} \end{bmatrix}, \qquad Y = \begin{bmatrix} Y_{11} & 0 \\ Y_{21} & Y_{22} \end{bmatrix} \tag{4}$$

where X_{11} and Y_{11} have dimension $t \times t$, then we call (X, Y) a proper generator.

Proper generators play a key role in the generalized Schur algorithm. The generalized Schur algorithm is a fundamental method for structured matrix factorizations.

A. Obtaining proper generators

For our purpose here we consider only the symmetric case.

Lemma 1: Consider a symmetric $n \times n$ matrix A with a block displacement equation (3). Then we can always find a Σ-unitary transformation matrix M such that XM is in proper form (4) and satisfies equation (3), i.e. $\nabla A = XM\Sigma M^T X^T$.

The constructive proof of this result is a block formulation of the example presented in the section 4.4.2 of [4]. This proof outlines an algorithm to obtain proper symmetric generators. See also section 2.5 of [6].

B. The Generalized Schur Algorithm

The generalized Schur algorithm computes the Schur complements of the leading $t \times t$ submatrices of A. There are several versions of this algorithm. See section 7.4 of [4] for the most general one. For our purposes we use the following result of [6].

Theorem 1: Let us assume that we have a proper symmetric generator X in form (4) which satisfies the displacement equation (3). Then the matrix

$$\bar{A} = A - \begin{bmatrix} X_{11} \\ X_{21} \end{bmatrix} [X_{11}^T \quad X_{21}^T] = \begin{bmatrix} 0_t & \\ & \bar{A}_{22} \end{bmatrix} \tag{5}$$

has t zero leading columns and rows. $\bar{A}_{22}$ is called the Schur complement of the $t \times t$ leading submatrix of A. A generator for $\bar{A}$ is given by $\bar{X} = [FX_{*1} \quad X_{*2}]$ where F is the same as in (3), i.e.

$$\bar{A} - F\bar{A}F^T = \bar{X}\Sigma\bar{X}^T. \tag{6}$$

Since F is strictly lower block triangular, namely, it has t zero leading rows, from (6) it is easy to see that if we eliminate the first t rows and columns of $\bar{A}$ and the first t rows of $\bar{X}$ we obtain the displacement equation $\bar{A}_{22} - \bar{F}\bar{A}_{22}\bar{F}^T = \bar{X}_2 \Sigma \bar{X}_2^T$, where $\bar{F}$ results from eliminating the first t rows and the first t columns of F. Now, if we can obtain a proper representation of $\bar{X}_2$ then we can obtain the Schur complement of the leading $t \times t$ submatrix of $\bar{A}_{22}$.

In the particular case when A is symmetric positive definite, this method can yield the Cholesky factorization of A. Consider $L = [L_1 \ L_2 \ \cdots \ L_\beta]$ such that

$$A = LL^T = L_1 L_1^T + L_2 L_2^T + \cdots + L_\beta L_\beta^T. \tag{7}$$

From equations (5) and (7) we can conclude that the first t columns of X are the first t columns of L. Then from Theorem 1, $\bar{A} = L_{t+1}L_{t+1}^T + L_{t+2}L_{t+2}^T + \cdots + L_\beta L_\beta^T$. So, if we obtain a proper representation $\tilde{X}$ of $\bar{X}$ we can see that

$$\begin{bmatrix} & 0_t & \\ \tilde{X}_{*1} & \cdots & \tilde{X}_{*t} \end{bmatrix}$$

are the next t columns of L.

III. Application of the Generalized Schur Algorithm

The LQ factorization and the reduction to the CEF of an $mk \times nl$ block Toeplitz matrix

$$T = \left[\begin{array}{ccc|ccc} T_0 & \cdots & T_{r-1} & T_r & \cdots & T_{n-1} \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ T_{-r+1} & \cdots & T_0 & T_1 & \cdots & T_{n-r} \\ \hline T_{-r} & \cdots & T_{-1} & T_0 & \cdots & T_{n-r-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ T_{-m+1} & \cdots & T_{-m+r} & T_{-m+r+1} & \cdots & T_{-m+n} \end{array}\right] \quad (8)$$

with $k \times l$ blocks T_i and where $r = \min(m,n)$, are direct applications of the generalized Schur algorithm described above.

A. Fast LQ factorization

Consider a block Toeplitz matrix T in form (8). We compute its LQ factorization $T = LQ$ as follows:

- Build the extended matrix

$$\bar{T} = \begin{bmatrix} TT^T & T \\ T^T & I \end{bmatrix} \quad (9)$$

  that is symmetric positive definite, and satisfies the block displacement equation

$$\nabla \bar{T} = \bar{T} - F\bar{T}F^T = X\Sigma X^T \quad (10)$$

  where $F = F_k \oplus F_l$ with

$$F_k = \begin{bmatrix} 0 & & & \\ I_k & 0 & & \\ & \ddots & \ddots & \\ & & I_k & 0 \end{bmatrix} \in \mathbb{R}^{mk\times mk}$$

$$F_l = \begin{bmatrix} 0 & & & \\ I_l & 0 & & \\ & \ddots & \ddots & \\ & & I_l & 0 \end{bmatrix} \in \mathbb{R}^{nl\times nl}.$$

- Compute the LQ factorization $[T_0 \cdots T_{n-1}] = L_0Q_0$ of the first block row of T, and build $[T_0 \cdots T_{n-1}] = \bar{L}_0C$ where $\bar{L}_0$ has full column rank $\bar{k} \leq k$ and $C = [C_0 \cdots C_{n-1}]$ contains the first $\bar{k}$ rows of Q_0.
- Define

$$\begin{bmatrix} Y_0 \\ \vdots \\ Y_{m-1} \end{bmatrix} = T \begin{bmatrix} C_0^T \\ \vdots \\ C_{n-1}^T \end{bmatrix}.$$

  Then the generator X of $\bar{T}$ satisfying displacement equation (10) with $\Sigma = I_{\bar{k}+l} \oplus -I_{\bar{k}+l}$ is given by

$$X = [X_1^T \quad X_2^T]^T \quad (11)$$

  where

$$X_1 = \begin{bmatrix} Y_0 & 0 & 0 & 0 \\ Y_1 & T_{-1} & Y_1 & T_{n-1} \\ Y_2 & T_{-2} & Y_2 & T_{n-2} \\ \vdots & \vdots & \vdots & \vdots \\ Y_{m-1} & T_{-m+1} & Y_{m-1} & T_{-m+n+1} \end{bmatrix},$$

$$X_2 = \begin{bmatrix} C_0^T & I_l & C_0^T & 0 \\ C_1^T & 0 & C_1^T & 0 \\ \vdots & \vdots & \vdots & \vdots \\ C_{n-1}^T & 0 & C_{n-1}^T & 0 \end{bmatrix}. \quad (12)$$

- Finally apply the generalized Schur algorithm with the generator X to obtain

$$\bar{T} = WW^T. \quad (13)$$

  It is easy to see that

$$W = \begin{bmatrix} L \\ Q^T \end{bmatrix}. \quad (14)$$

  Moreover, we can verify that if w_{ij} is the leading entry of the jth column of L for $j = 1, 2, \ldots, nl$, then the row T_{i*} is linearly independent from the previous rows.

B. Fast Column Echelon Form

The CEF form of a matrix T is equivalent to the triangular part L of its LQ factorization $T = LQ$. From (13) and (14) notice that the first mk rows of W are equal to L. From (11) and (12) notice also that the first mk rows of the generator X are given by X_1. So, if we apply the fast LQ method using only $X = X_1$ then we obtain only the column echelon form of T. Notice that considering matrix X_2 in relation (11) is equivalent to computing the orthogonal matrix Q.

IV. Block displacement structure algorithms

For the algorithms presented here we consider that the rank ρ of $A(s)$ is given. There are several documented algorithms to obtain the rank of a polynomial matrix, see for example [3]. See also [11] where we show how we can obtain the rank of a polynomial matrix while obtaining its eigenstructure.

A. The fast LQ algorithm

Notice that matrix R_i in algorithm `LQ` of [12] is a particular case of matrix (8) where $m = d + i$, $n = i$ and where k and l are the dimensions of the polynomial matrix $A(s)$. So we can reduce matrix R_i by applying the fast LQ factorization. Because of the particular form of R_i, its generator matrix can be obtained from the generator of R_{i-1} as follows. Consider the LQ factorization

$$A_0 = [\Delta_0 \quad 0] \begin{bmatrix} M_1^T \\ N_1^T \end{bmatrix}.$$

A generator of R_1 is then given by $G_1 = [X_1^T \quad Y_1^T]^T$ where

$$X_1 = \left[\boxed{\begin{matrix} \Delta_0 & 0 & 0 \\ A_1M_1 & A_1 & A_1M_1 \\ A_2M_1 & A_2 & A_2M_1 \\ \vdots & \vdots & \vdots \\ A_dM_1 & A_d & A_dM_1 \end{matrix}} \; \boxed{\begin{matrix} 0 \\ A_0 \\ A_1 \\ \vdots \\ A_d \end{matrix}} \right],$$

$$Y_1 = \left[\begin{matrix} M_1 & I_l & M_1 & 0 \end{matrix}\right].$$

At the following step, the LQ factorization of the first block row of R_2 is given by

$$[A_0 \quad 0] = [\Delta_0 \quad 0 \quad 0] \left[\begin{array}{c|c} M_1^T & 0 \\ N_1^T & 0 \\ \hline 0 & \times \end{array}\right],$$

so the corresponding generator is given by $G_2 = [X_2^T \quad Y_2^T]^T$ where

$$X_2 = \left[\begin{matrix} \boxed{\begin{matrix} \Delta_0 & 0 & 0 \\ A_1M_1 & A_1 & A_1M_1 \\ A_2M_1 & A_2 & A_2M_1 \\ \vdots & \vdots & \vdots \\ A_dM_1 & A_d & A_dM_1 \end{matrix}} \\ 0 \end{matrix} \; \begin{matrix} 0 \\ \boxed{\begin{matrix} 0 \\ A_0 \\ A_1 \\ \vdots \\ A_d \end{matrix}} \end{matrix} \right],$$

$$Y_2 = \left[\begin{matrix} M_1 & I_l & M_1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}\right].$$

In summary, with the fast LQ method we can also use some of the information obtained at the step i in the computations at step $i+1$.

Algorithm `fastLQ`
For a $k \times l$ polynomial matrix $A(s)$ of degree d and rank ρ, this algorithm computes the vectors of a minimal basis of its null-space.

- Step 1:
  Compute the generator $G_1 = [X_1^T \quad Y_1^T]^T$ of R_1 and partition $X_1 = [X_{1_1} \quad X_{1_2}]$ where X_{1_2} contains the last l columns of X_1.
  Apply the generalized Schur algorithm with G_1 in order to obtain the LQ factorization $R_1Q_1 = L_1$. If $\gamma_1 = \mu = l - \operatorname{rank} R_1 \neq 0$, then extract the μ vectors of degree 0 of the basis of the null-space of $A(s)$ from the last μ columns of Q_1.
- Step i:
  Build the generator $G_i = [X_i^T \quad Y_i^T]^T$ where
  $$X_i = \left[\begin{matrix} X_{(i-1)_1} & 0 \\ 0 & X_{(i-1)_2} \end{matrix}\right], \quad Y_i = \left[\begin{matrix} Y_{i-1} \\ 0 \end{matrix}\right],$$
  Apply one step of the generalized Schur algorithm, namely, process the first k rows of G_i in order to obtain a proper generator $\bar{G}_i = [\bar{X}_i^T \quad \bar{Y}_i^T]^T$. Partition $\bar{X}_1 = [X_{1_1} \quad X_{1_2}]$ where X_{1_2} contains the last l columns of $\bar{X}_1$.
  Apply the generalized Schur algorithm with generator $\bar{G}_i$ to obtain the LQ factorization $R_iQ_i = L_i$. Obtain $\gamma_i = li - \operatorname{rank} R_i - \sum_{j=1}^{i-1} \gamma_j$. If $\gamma_i - \gamma_{i-1} = \mu \neq 0$ then extract the μ vectors of degree $i-1$ of the basis of the null-space of $A(s)$ from the last μ columns of Q_i.
- Stopping rule:
  When $\gamma_i = l - \rho$ we have all the vectors of a minimal basis of the null-space of $A(s)$.

B. The fast CEF algorithm

Notice that matrix $\bar{R}_j$ analyzed at each step in algorithm `CEF` of [12] is a particular case of matrix (8) where $m = j$, $n = d + j$, and where k and l are the dimensions of the polynomial matrix $A(s)$. So we can compute the CEF of matrix $\bar{R}_j$ as described in section 3.

Algorithm `fastCEF`
For a $k \times l$ polynomial matrix $A(s)$ of degree d and rank ρ, this algorithm computes the vectors of a minimal basis of its null-space.

- Initialization:
  Compute $d_{\max}$, the maximum expected null-space degree, from equation (9) in [12]. Start with $j = 1$ and $\Delta = 0$.
- Step i:
  With the fast CEF method described in section 3, compute the CEF of $\bar{R}_j$. Derive the number μ of linearly dependent columns of $A(s)$.
  Update $\Delta = \Delta + \frac{d_{\max}}{10}$ and then $j = j + \Delta$.
- Stopping rule:
  When $\mu = l - \rho$ we can find from $\bar{R}_f$, the last CEF calculated, all the vectors in the basis of the null-space of $A(s)$.
- Obtaining the null-space:
  Compute the linear combination of the linearly dependent rows of $\bar{R}_f$ by solving the corresponding linear triangular system as in equation (5) of [12]. If more than one row corresponds to the same linearly dependent column in $A(s)$, take the combination of the smallest degree.

V. Performance analysis

Several numerical tests have been carried out to analyze the performance of the algorithms `fastLQ` and `fastCEF`. The algorithms have been programmed in Matlab using the Polynomial Toolbox [10] on a Sun Ultra 5 work-station. The algorithms have been implemented from scratch, without using Matlab built-in functions such as `qr` or `rref`.

A. Algorithmic complexity

In the first part of this work [12] we have seen that the classical LQ and CEF algorithms, which process block Toeplitz matrices of η columns at each step, have complexity $O(\eta^3)$. In this paper we have analyzed the fast LQ method and the fast CEF method. The characteristic of these fast methods is that they process the columns of the generator matrix X satisfying the displacement equation (3). Generator X has $\alpha = p + q$ columns. So, considering a square $\eta \times \eta$ block Toeplitz matrix, it can be showed that if $\alpha \ll \eta$, then the complexity of fast methods LQ and CEF can be considered as $O(\eta^2)$ [5]. In brief, for very large polynomial matrices, by applying fast algorithms `fastLQ` and `fastCEF`, we can expect a gain of one order of magnitude in algorithmic complexity. Nevertheless, a deepest analysis must be done to determine if the fast algorithms are really more efficient, in practice, than classical algorithms `LQ` and `CEF` of [12].

First we analyze the performance of fast methods LQ and CEF applied on a non-square block Toeplitz matrix T in form (8). In section 3 we saw that the fast CEF method is the application of the fast LQ method with a shorter generator. So, here we only analyze the fast LQ method.

The fast LQ method achieves its maximal performance, in comparison with the classical method, when processing a generator with a small number of columns with respect to the number of columns of T. The number of columns of matrix T in form (8) grows with the number of columns of its blocks or with the number of block columns. On the other hand, the number of columns of the generator given by (11) and (12) decreases if the rank of the first block row of T is small. The rank of the first block row of T is less than or equal to $\min(k, l)$. In conclusion, in order to assure the best performance of the fast method we have to apply it onto block Toeplitz matrices in form (8) with $l \geq k$ moderately large and with n large. Notice that the number of block rows m has the same influence in both classical and fast methods.

In figure 2 we present the execution times of the LQ method and the fast LQ method applied onto a block Toeplitz matrix in form (8) with $k = 20$, $m = 10$, $n = 7$ and $l = 2, 3, \ldots, 35$. Notice that the fast method achieves its best performance when $l \geq 20$. The value of l from which the fast method begins being more efficient than the classical method is different for each matrix T and seems to be difficult to predict. Nevertheless we can see that if n is larger, then the fast method is more efficient than the classical method for smaller values of l. If we take the same matrix used in figure 2 but with $n = 10$, we obtain figure 3. Notice that the fast method is more efficient even when $k \geq l$.

Now we analyze the performance of algorithm `fastLQ`. Consider a $k \times l$ polynomial matrix $A(s)$ with degree d. The total amount of elementary operations performed by the algorithm is equal to the sum of elementary operations performed at each step. Therefore, the final execution time is the sum of the execution times needed to obtain the null-spaces of the different Toeplitz matrices R_i. In figure 4 we present the execution times of the LQ method and the fast LQ method applied onto a block Toeplitz matrix in form (8) with $k = 18$, $l = 18$, $m = 2 + n$, and $n = 3, 4, \ldots, 17$. This corresponds to the application of algorithms `LQ` and `fastLQ` onto a 18×18 polynomial matrix of degree 2 which has at least one vector of degree 16 in the basis of

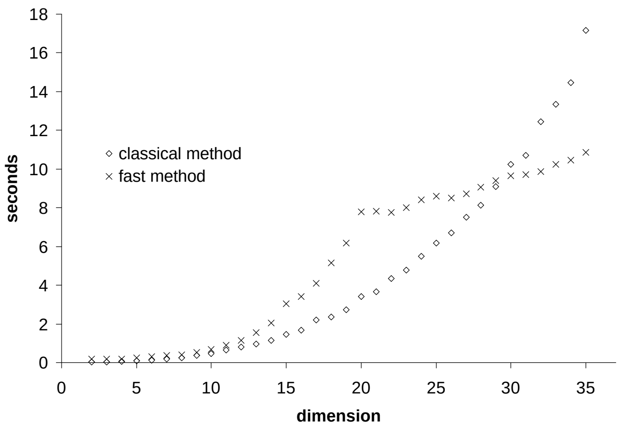

Fig. 2. Execution times for different numbers of columns in the blocks of a block Toeplitz matrix of 7 block columns.

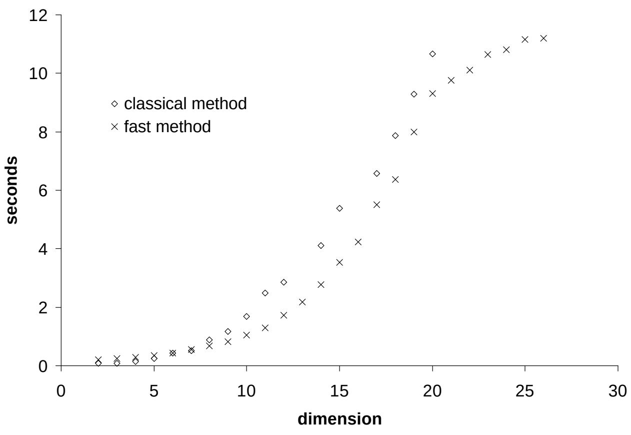

Fig. 3. Figure 3: Execution times for different numbers of columns in the blocks of a block Toeplitz matrix of 10 block columns.

its null-space. Notice that even if the classical LQ method is more efficient when $n \leq 10$, the total execution time of algorithm `LQ` is larger than that of algorithm `fastLQ`. Graphically we can consider that the total execution times are the surfaces below the curves in figure 4.

Obviously we can also find polynomial matrices for which the classical algorithm `LQ` is more efficient than its fast version. Consider for example a 18×18 polynomial matrix of degree 2 and consider that it has a null-space of maximal degree equal to 6. From figure 4 we can see that the sum of the execution times for $n = 3, 4, 5, 6, 7$ is larger for the fast method than for the classical method.

The same analysis can be applied for algorithms `CEF` and `fastCEF`. In this case the degree d of the polynomial matrix $A(s)$ has a direct influence on the number of columns of the block Toeplitz matrices $\bar{R}_j$. So, we can find examples where, even if the degree of the null-space is not large, with a large degree d we can assure that the fast algorithm

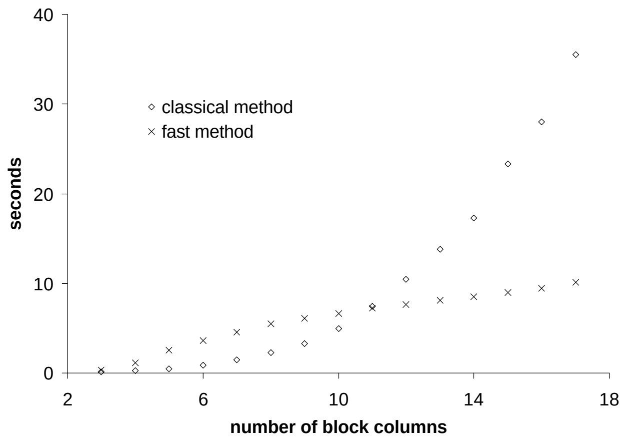

Fig. 4. Figure 4: Execution times for different numbers of block columns in a block Toeplitz matrix.

TABLE I

EXECUTION TIMES IN SECONDS WHEN APPLYING THE FOUR POLYNOMIAL NULL-SPACE EXTRACTION ALGORITHMS TO THE MASS-SPRING EXAMPLE WITH p MASSES.

p	CEF	fastCEF	LQ	fastLQ
2	0.09	0.08	0.06	0.11
3	0.10	0.15	0.08	0.16
4	0.11	0.21	0.10	0.26
5	0.18	0.34	0.15	0.38
6	0.26	0.44	0.19	0.44
7	0.43	0.61	0.24	0.63
8	0.81	0.82	0.33	0.72
9	1.53	1.08	0.44	1.05
10	2.75	1.60	0.57	1.20
11	3.91	2.12	0.73	1.73
12	6.29	2.86	0.97	1.97
13	10.23	3.87	1.52	3.31
14	16.82	5.47	1.69	3.51
15	27.11	7.21	1.74	4.94

is more efficient than the classical algorithm. Finally we present a numerical example.

Consider the mass-spring system of figure 1 with $k_i = m_i = 1$ for $i = 2, 3, \ldots, 15$, on which we apply the four algorithms to solve system (2). We present the execution times of each algorithm in Table 1. Notice the important difference between algorithms `CEF` and `fastCEF`. For the kind of polynomial matrices involved in (2), the fast method CEF is more efficient even when p is not so large. Nevertheless, algorithms `LQ` and `fastLQ` are the best choice to solve this kind of problems. Notice also that in this case, the fast method LQ does not achieve its best performance and thus, algorithm `LQ` is more efficient than algorithm `fastLQ` even when p is large. In order that algorithm `fastLQ` be more efficient than algorithm `LQ`, it needs to perform more steps. In other words, for a same number of masses, a polynomial matrix $[D(s)\ -B]$ having a null-space of degree greater than $2p$ would be needed. From the equation (9) in [12], which gives the maximum expected null-space degree, we can see that this is not possible.

B. Numerical stability

In the first part of this work [12], we concluded that even if the backward-stability of algorithms `LQ` and `CEF` is not guaranteed, they proved to be reliable in almost all the practical cases. A similar conclusion can be drawn about the stability of algorithms `fastLQ` and `fastCEF`.

If the implementation of the generalized Schur algorithm is done considering the four enhancements proposed in section 2.5 of [5], then, its backward-stability can be guaranteed by working with symmetric positive definite Toeplitz matrices. In order to ensure positive definiteness, fast methods LQ and CEF use the extended matrix (9) and the displacement equation (10). Nevertheless, the backward-stability of the generalized Schur algorithm using the extended matrix (9) is not proven. In fact, the computed matrix $\bar{Q}$ obtained as the result of the fast method LQ can be non-orthogonal [5, section 3.2]. In other words, we cannot ensure the backward-stability of fast methods LQ or CEF but only a *weak stability*, namely, if $A = LQ$ is the exact LQ factorization of Toeplitz matrix A and $\bar{L}$ and $\bar{Q}$ are the computed matrices, then we can show that norms $\|AA^T - \bar{L}\bar{L}^T\|$ and $\|I - \bar{Q}\bar{Q}^T\|$ are small. See chapter 4 of [5] for a deepest analysis and the definition of weak stability.

In summary, we cannot ensure the backward stability of algorithms `fastLQ` and `fastCEF`. Nevertheless, we claim that, in practice, they perform well for most of the polynomial matrices arising in control problems.

VI. CONCLUSIONS

In this paper, we have proposed two algorithms based on fast factorization methods. The improvement brought by these fast methods, when they are applied onto very large Toeplitz matrices, is evident. We can expect a gain of one order of magnitude in the computational complexity of the fast algorithms LQ and CEF with respect to the algorithmic complexity of the classical algorithms presented in [12]. Nevertheless, applying systematically the fast algorithms is not always the best option. We can always find polynomial matrices in relevant control problems for which the classical algorithms are more efficient than their fast versions, see Table 1.

Regarding numerical stability, we have seen that while fast algorithms are faster, stability and robustness guarantees are weaker. Nevertheless, in practice, we experienced that the algorithms `fastLQ` and `fastCEF` give reliable results even if we cannot ensure their backward-stability.

From the point of view of numerical analysis, it seems that a trade-off must be found between computational speed and stability and robustness. In order to ensure stability we have to perform a larger number of elementary operations. On the other hand, if we want to diminish the execution time we have to lose accuracy. A good compromise must be established depending on the application of the algorithm.

On the one hand, ensuring backward stability at any cost may not always be the best choice. For example, in the case of our algorithms, we do not need to use the singular value decomposition since by using the cheaper LQ factorization, we obtained satisfactory results. Ensuring weak stability [5, section 4.2] of an algorithm can be sufficient. On the other hand, having always as an objective diminishing the order of magnitude of the algorithmic complexity, even allowing a lost of accuracy, does not guaranteed a better performance. In this paper we have described a relevant control problem for which the classical methods still prove more efficient.

ACKNOWLEDGMENTS

Juan-Carlos Zúñiga acknowledges support from the National Council of Science and Technology of Mexico (CONACYT) as well as from the Secretariat of Public Education of Mexico (SEP). Didier Henrion acknowledges support from project No. 102/02/0709 of the Grant Agency of the Czech Republic as well as from project No. ME 698/2003 of the Ministry of Education of the Czech Republic.

REFERENCES

[1] Datta B.N., Elhay S. and Ram Y.M., Orthogonality and Partial Pole Assignment for the Symmetric Definite Quadratic Pencil. *Linear Algebra and Its Applications,* Vol. 257, pp. 29-48, 1997.

[2] Golub G.H. and Van Loan C.F., *Matrix Computations.* The Johns Hopkins University Press, New York, 1996.

[3] Henrion, D. and Šebek, M., Numerical methods for polynomial rank evaluation. *Proc. of the IFAC Symposium on System, Structure and Control*, Nantes, France, 1998.

[4] Kailath T. and Sayed A. H., Displacement Structure: theory and applications. *SIAM Review*, vol. 37, pp. 297-386, 1995.

[5] Kailath T. and Sayed A. H., *Fast Reliable Algorithms for Matrices with Structure.* SIAM, Philadelphia, 1999.

[6] Kressner D., *Numerical Methods for Structured Matrix Factorizations.* Ph.D. thesis, Technical University of Chemnitz, Germany, June 2001.

[7] Kressner D. and Van Dooren P., *Factorizations and linear system solvers for matrices with Toeplitz structure*. SLICOT Working Note 2000-2, June 2000.

[8] Megretski A., *Lecture notes on multivariable control systems*. MIT, Cambridge, MA, 2000.

[9] NICONET, *The Control and System Library SLICOT*. See `www.win.tue.nl/niconet`.

[10] Polyx, Ltd. *The Polynomial Toolbox for Matlab.* Version 2.5 released in 2000. See `www.polyx.cz`

[11] Zúñiga J.C. and Henrion D., Block Toeplitz Methods in Polynomial Matrix Computations. *To be registered as a LAAS-CNRS research report. Submitted to the Symposium on Mathematical Theory of Networks and Systems*, Leuven, Belgium, July 5-9, 2004.

[12] Zúñiga J.C. and Henrion D., On the application of different numerical methods to obtain null-spaces of polynomial matrices. Part 1: block Toeplitz algorithms. *To be registered as a LAAS-CNRS research report. Submitted to the IEEE Conference on Decision and Control*, Paradise Island, Bahamas, December 14-17, 2004.