BANDED MATRIX SOLVERS AND POLYNOMIAL DIOPHANTINE EQUATIONS *

Martin Hromčík[†], *Zdeněk Hurák*[†],
Michael Šebek[♯] *and Radek Frízel*[♯]

[†] Institute of Information Theory and Automation
182 08 Prague, Czech Republic
{hromcik,hurak}@utia.cas.cz

[‡] Centre for Applied Cybernetics, Faculty of Electrical Engineering,
Czech Technical University in Prague

[♯] Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague

Abstract

Numerical procedures and codes for linear Diophantine polynomial equations are proposed in this paper based on the banded matrix algorithms and solvers. Both the scalar and matrix cases are covered. The algorithms and programs developed are based on the observation that a set of constant linear equations resulting from the polynomial problem features a special structure. This structure, known as Sylvester, or block Syelvester in the matrix case, can in turn be accommodated in the banded matrix framework. Reliable numerical algorithms and programs for banded matrices are readily available at present, for instance in the well known LAPACK package. Software routines based on dedicated LAPACK band matrix solvers were programmed in the C language and linked to MATLAB, mainly for two reasons: to provide prospective users with an environment most of them are familiar with, and to gain the possibility of direct comparison with related functions of the Polynomial Toolbox for MATLAB. Performance of the codes was evaluated by extensive numerical experiments and also in a real-life audio application.

1 INTRODUCTION

Polynomial Diophantine equations play a crucial role in the polynomial theory of control systems synthesis. The origin of the polynomial or algebraic approach is dated to the early 70s. Systems are described by input-output relations, similarly to the classical control techniques, however, the transfer functions are not regarded as functions of complex variable but as algebraic objects. The design procedures are then reduced to algebraic operations with polynomial matrices, typically to solving algebraic polynomial equations. This approach not only enables to resolve many existing control problems in a more elegant and unifying way but also provides further insight into the structure of the control systems and shows new relationships between various control tasks. An interested reader is referred to the textbooks [1, 2].

Almost all polynomial design routines lead to linear Diophantine equations in the form

$$A(s)\,X(s) + B(s)\,Y(s) = C(s), \tag{1}$$

*The work of Martin Hromčík and Zdeněk Hurák was supported by the Grant Agency of The Czech Republic under the contract No. 102/05/0011. The work of Radek Frízel was supported by the Ministry of Education of the Czech Republic under contract No. 1M6840770004. The work of Michael Šebek was supported by the Grant Agency of The Czech Republic under the contract No. 102/05/2075.

where $A(s)$, $B(s)$ and $C(s)$ are polynomials, or polynomial matrices in the multi-input and/or multi-output (MIMO) configuration. Applications include closed-loop pole placement [3], minimum variance control [4], LQ and LQG optimal compensators [1], adaptive and predictive control [4], or time optimal controllers [1, 3].

Let us recall the basic facts of the theory of polynomial Diophantine equations in the following. An interested reader can consult for instance the textbooks [1, 2] for further subtleties.

The Diophantine equation in the scalar case is solvable if and only if every common divisor of $A(s)$ and $B(s)$ is a divisor of $C(s)$. Since the Diophantine equation is linear any and all solutions pairs of (1) are given by

$$\begin{aligned} X(s) &= X_0(s) + B(s)\,Q(s), \\ Y(s) &= Y_0(s) - A(s)\,Q(s), \end{aligned} \tag{2}$$

where

$$Q(s) = Q_0 + Q_1\,s + Q_2\,s^2 + \cdots$$

is an arbitrary polynomial.

Polynomial matrix Diophantine equations can be handled alike. In general, if $A(s)$, $B(s)$ and $C(s)$ are matrices of dimensions m x p, m x k and m x q, respectively, then the equation (1) has a solution $\{\ X_0(s),\ Y_0(s)\ \}$ if and only if any common divisor of $A(s)$ and $B(s)$ is a left divisor of C. Any and all solutions pairs of Diophantine equation (1) in the matrix case are then given by

$$\begin{aligned} X(s) &= X_0(s) + \tilde{B}(s)Q(s), \\ Y(s) &= Y_0(s) - \tilde{A}(s)Q(s), \end{aligned} \tag{3}$$

where

$$A^{-1}(s)B(s) = \tilde{B}(s)\tilde{A}^{-1}(s)$$

and $Q(s)$ is an arbitrary polynomial matrix of appropriate dimensions.

Since matrix multiplication is not commutative, there also exists the *dual* polynomial matrix equation in the form

$$X(s)\,A(s) + Y(s)\,B(s) = C(s), \tag{4}$$

Note that in the scalar case these two cases coincide thanks to comutativeness of scalar product.

2 Algorithms for Diophantine equations

A particular solution to a given polynomial Diophantine equation can be found through interpolation, symbolic procedures, or the Sylvester matrix approach.

2.1 Interpolation

Interpolation is a useful tool for handling polynomials and polynomial matrices. It can be used both for evaluation of various functions of polynomial matrices (e.g. products, scalar power, determinant, adjoint, etc.) and for polynomial equations and other more complicated problems encountered in control.

The basic idea is as follows: The input polynomials are evaluated at suitably chosen points at first. In this way, an equivalent representation of the problem in terms of constants is obtained. The computation is then performed within these constants and the desired solution is finally recovered, typically by solving a Vandermonde or generalized Vandermonde linear system.

Interpolation-based results for some control problems involving polynomials and polynomial matrices, including the Diophantine equations, have been given in the survey paper [5].

2.2 Symbolic procedures

Polynomials can be represented by their coefficients which are finite-precision numbers. In principle it is also possible to represent polynomial matrices as symbolic entities and to employ symbolic computation tools (e.g. Symbolic Math Toolbox for MATLAB [6], MATHEMATICA [7]). However, this approach suffers from rather severe difficulties. A great problem with symbolic routines is their computational time consumption. The methods are rather costly even for small-size problems and, what is worse, the execution time increases usually exponentially with the size of the task. The efficiency of the methods also depends on data types - for instance they are much faster for polynomials with integer entries and extremely slow for coefficients with many decimal digits. All number are handled without rounding which also affects large memory requirements of such algorithms. These limitations make this approach practically unusable for the purposes of control systems design.

2.3 Sylvester matrix methods

Many computations with polynomials and polynomial matrices can be expressed in terms of related Sylvester matrices. This interpretation is straightforward and leads to the set of constant linear equations the numerical methods of which are well understood.

To illustrate this approach let us consider a simple polynomial equation of the type

$$A(s)\,X(s) = B(s), \tag{5}$$

where

$$\begin{aligned} A(s) &= A_0 + s\,A_1 + ... + s^{d_A}\,A_{d_A} \\ B(s) &= B_0 + s\,B_1 + ... + s^{d_B}\,B_{d_B} \\ X(s) &= X_0 + s\,X_1 + ... + s^{d_X}\,X_{d_X} \end{aligned}$$

Comparing respective powers of s in the above matrix polynomial equation one can build an equivalent linear system of equations

$$\underbrace{\begin{bmatrix} A_0 & & & 0 \\ A_1 & A_0 & & \\ \vdots & A_1 & \ddots & \\ A_{d_A} & \vdots & & A_0 \\ & A_{d_A} & & A_1 \\ & & \ddots & \vdots \\ 0 & & & A_{d_A} \end{bmatrix}}_{\langle A \rangle_{d_X}} \underbrace{\begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{d_X} \end{bmatrix}}_{\overline{X}} = \underbrace{\begin{bmatrix} B_0 \\ B_1 \\ \vdots \\ B_{d_B} \end{bmatrix}}_{\overline{B}}, \tag{6}$$

Matrix $\langle A_{d_X} \rangle$ is referred to as the Sylvester matrix of $A(s)$ of order d_X if $A(s)$ is a polynomial, or block Sylvester matrix if $A(s)$ is a polynomial matrix.

This approach can be easily adopted also for our Diophantine equation (1). Using the notation of (6), one can write

$$\left[\langle A \rangle_{d_X}\ \langle B \rangle_{d_Y}\right] \begin{bmatrix} \overline{X} \\ \overline{Y} \end{bmatrix} = \overline{C}$$

2.4 Algorithm recommendation

To sum up, the symbolic approach is hardly usable for medium-size problems and completely unusable for large data whereas both the interpolation and Sylvester matrix results can be employed in these situations. In addition, as evidenced in the PhD. thesis [8], the Sylvester methods appear more efficient and lead to better conditioned linear systems. Simply put, the interpolation approach can be decomposed into a procedure related to the choice of interpolating points

and a subsequent Sylvester-like set of equations (see [8], pp. 70, for a more rigorous reasoning). For this reason, almost all software tools for polynomial equations, including the Polynomial Toolbox for MATLAB [9], PolPack++ [10], or the Polynomial Package for MATHEMATICA [11], are based on the Sylvester matrix approach.

The Sylvester matrix methods are elegant and numerically efficient, nevertheless, several problematic issues arise. For instance, the estimation of the solution's degree is rather tricky in the matrix case. This is however a key parameter influencing the size of the Sylvester matrix and, consequently, the number of equations and unknowns in the equivalent linear system of equations (6). Basically, an upper and lower bounds on d_X can be found based directly on the coefficients of $A(s)$, $B(s)$ and $C(s)$. The minimum-degree solution is then sought after by a standard or modified binary search.

In addition, the sizes of the Sylvester matrices are usually rather high and the method when implemented require a large amount of memory. This drawback can certainly be reduced if the structure of Sylvester matrices is considered.

3 Structure of Sylvester and block Sylvester matrices

Sylvester and block Sylvester matrices feature a strong structure. Large Sylvester matrices are based on only relatively few independent coefficients of underlying polynomials or polynomial matrices. Although dedicated solvers for this structure have been a subject of intensive research recently, see for instance the reports [12, 13], these routines are far from being standard today and are not implemented in respected software libraries at present. This conclusion is also reflected by the fact that all the packages involving polynomial equations mentioned above rely on standard general linear set solvers, ignoring the Sylvester structure completely.

However, one can easily recognize that the Sylvester matrices can be fitted into the banded matrices framework easily, to exploit their structure partly at least. We decided to explore this idea, suggested and implemented related solvers using the well-established LAPACK library linked to the well known MATLAB environment, and proved the usefulness of this approach by numerical experiments and in a real-life case study. Details are given in the following sections.

4 LAPACK

LAPACK [14] is a program library for linear algebra problems written in Fortran 77. It provides routines for solving systems of linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorisations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also handled, as well as related computations such as reordering of the Schur factorisations and estimation of condition numbers. Next to general dense matrices, routines for banded matrices are also provided which is most important for our purposes. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision. LAPACK routines are written in the way that the greatest possible portion of computations is delegated to the efficient and computer-optimized Basic Linear Algebra Subprograms (BLAS) [15].

The original goal of the LAPACK project was to make the widely used EISPACK [17] and LINPACK [16] libraries run efficiently on shared-memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multi-layered memory hierarchies of the machines, thereby spending too much time moving data instead of doing useful floating-point operations. LAPACK addresses this problem by reorganizing the algorithms to use block matrix operations, such as matrix multiplication, in the innermost loops. These block operations can be optimized for each architecture to account for the memory hierarchy, and so provide a transportable way to achieve high efficiency on diverse modern machines.

LAPACK is a freely available software package provided on the World Wide Web via netlib, anonymous ftp, and http access [14]. In addition, its routines have been adopted by commercial

software providers and even the leading computational tools including MATLAB and MATHEMATICA rely on LAPACK when solving sets of linear equations, finding eigenvalues or factoring matrices.

4.1 MATLAB and LAPACK

Many MATLAB's functions are essentially just calls to LAPACK routines. Unfortunately, functions for banded matrices are not among them and an interface between MATLAB and LAPACK has to be programmed.

For using LAPACK's functions in MATLAB, the external interface in the form of MEX-files can be employed, see [18]. These files can be written in the programming languages C or Fortran, forming a gate between MATLAB and functions from the LAPACK library. Calling LAPACK library within MATLAB is well described for example in the MATLAB Help or in [18].

5 Implemented functions

A MATLAB function `axbycL` was written based on the LAPACK's banded matrix routines. The function accepts both scalar polynomial and polynomial matrices. MATLAB version 6 or higher is required, the polynomial Toolbox for MATLAB version 2.5 is recommended for user's comfort (it is not essential, however). The routine together with auxiliary MATLAB-LAPACK interface procedures and basic documentation can be downloaded at
`http://ar.c-a-k.cz/hromcik/diophantine.zip`

Our function finds a particular solution $[X_0, Y_0]$ of the polynomial and matrix polynomial Diophantine equation $A\,X + B\,Y = C$. If no polynomial solution exists then all the entries of $[X_0\,,\,Y_0]$ are set to NaN.

In the scalar case, the Sylvester matrix is square, for instance

$$S = \begin{bmatrix} A_0 & 0 & 0 & 0 & B_0 & 0 & 0 \\ A_1 & A_0 & 0 & 0 & B_1 & B_0 & 0 \\ A_2 & A_1 & A_0 & 0 & B_2 & B_1 & B_0 \\ 0 & A_2 & A_1 & A_0 & B_3 & B_2 & B_1 \\ 0 & 0 & A_2 & A_1 & 0 & B_3 & B_2 \\ 0 & 0 & 0 & A_2 & 0 & 0 & B_3 \end{bmatrix}$$

It is matched to a square band matrix of the type

$$\begin{bmatrix} xx & xx & xx & xx & xx & 0 & 0 \\ xx & xx & xx & xx & xx & xx & 0 \\ xx & xx & xx & xx & xx & xx & xx \\ 0 & xx & xx & xx & xx & xx & xx \\ 0 & 0 & xx & xx & xx & xx & xx \\ 0 & 0 & 0 & xx & xx & xx & xx \end{bmatrix}$$

that can be passed to LAPACK, namely to its solver for banded set of linear equations `dgbsv`, see [14].

Situation gets slightly more complicated for polynomial matrices. First, the minimum degree of the solution cannot be estimated easily in advance and is sought after by a binary-search procedure described in the introductory section. Alternatively, the user can specify the degree as an optional input parameter if he knows in advance that such a solution exists. In addition, the LAPACK's `DGBSV` banded solver does not accept rectangular block Sylvester matrices which is typical in the polynomial matrix case. Fortunately, banded matrix factorization routines for LU factorization can be employed instead together with a trapezoidal linear set solver. More specifically, LAPACK functions `dgbtrf` and `dtzrzf` were addressed, see [14].

6 Numerical testing

Numerical experiments have been carried out to evaluate efficiency of the implemented function `axbycL`. The function was compared to the Polynomial Toolbox for MATLAB function `axbyc` with similar functionality [9].

The band matrix solvers included in LAPACK are based on numerically stable algorithms [14]. The implemented function `axbycL` hence achieves accuracy comparable to the Polynomial Toolbox routine `axbyc`, based on general matrix MATLAB's and, consequently, LAPACK's solvers in fact [18]. This observation applies in the case of generic well-conditioned data as well as for some specific problematic instants - for instance, with almost non-coprime polynomials $A(s)$ and $B(s)$.

The implemented function `axbycL` is however significantly faster than the Polynomial Toolbox as illustrated further. All computations were performed on an IBM PC computer with Intel Pentium IV processor, 800 MHz, and 1024 MB RAM. Software versions were 6.5 for MATLAB, 3.0 for LAPACK and 2.5 for the Polynomial Toolbox. Test polynomials coefficients were generated as random numbers ranging the interval $[0, 1]$. Resulting times were computed as averages of ten independent runs for each degree and size combination. Some results for scalar polynomials and polynomials matrices are presented in the figures 1 and 2.

[scale=.5]Test3$_2$

Figure 1: Comparison of Polynomial toolbox's function axbyc and our function axbycL for scalar polynomial Diophantine equation.

[scale=.5]Test4$_2$

Figure 2: Comparison of Polynomial toolbox's function axbyc and our function axbycL for matrix polynomial Diophantine equation. Square polynomial matrices of degree 10 were used.

The suggested approach is especially effective compared to the available solvers in the case of high-degree scalar polynomials. One might argue that this particular case is not of special interest as we rarely cope with systems of orders of several hundreds. However, even such polynomials are useful in real life applications, namely in the field of signal processing. We present and audio-improvement case study in the sequel as an illustration of this statement.

7 Upgrading Loudspeakers Dynamics

An original approach has been published by Sternad et al. in [19] to improve performance of an audio equipment at low additional costs. The authors use the LQG optimal feedforward compensator technique to receive an inverse dynamic filter for a moderate quality loudspeaker. By attaching a signal processor implementing this filter prior to the loudspeaker, the dynamical imperfections of the original device are eliminated and the overall equipment behaves as an aparatus of a much higher class.

Unlike their predecessors, the authors try to modify the sound over the whole range of frequencies. Such a complex compensation fully employs the increasing performance of signal hardware dedicated to CD-quality audio signals, and at the same time calls for fast and reliable polynomial equations solvers [19].

The loudspeaker dynamics is considered in the form of an ARX model

$$y(t) = z^{-k}\frac{B(z)}{A(z)}u(t).$$

Since the impulse response is rather long for a high sampling frequency (CD-quality standard of 44 kHz was used), both the numerator and denominator of the model are of high orders, say one to five hundred.

The model has an unstable inverse in general since some of its zeros may lie outside the unit disc. Hence a stable approximation has to be calculated to be used in the feedforward structure. The authors recall the LQG theory and seek for a compensating filter

$$u(t) = \frac{Q(z)}{P(z)} w(t)$$

such that the criterion

$$J = E\left[|y(t) - w(t-d)|^2 + \rho |u(t)|^2\right]$$

is minimized.

For broadband audio signals, the optimal filter is given in the form

$$u(t) = \frac{Q_1(z)A(z)}{\beta(z)} w(t)$$

where β results from the spectral factorization

$$\beta\beta^* = BB^* + \rho AA^*$$

and Q_1 is the solution of a subsequent Diophantine equation

$$q^{k-d} B^*(q) = r\beta^*(q) Q_1(q^{-1}) + qL^*(q), \tag{7}$$

see [19].

Let us perform a benchmark experiment to compare the existing approach and our newly proposed algorithm for particular numerical data kindly provided by Mikael Sternad and colleagues from the University of Uppsala. The data in concern are given as follows. The numerator $B(z) = B_0 + B_1 z^{-}1 + \cdots + B_{250} z^{-250}$ is an unstable polynomial of degree 250, $A(z)$ is stable of degree 90, and $k = 160$. Taking $\rho = 0$, the spectral factorization of $m(z) = B(z)B^*(z) = m_{250} z^{-250} + \ldots + m_0 + \ldots + m_{250} z^{250}$ is performed first, using the Polynomial Toolbox for MATLAB function `spf`.

Solution of the Diophantine equation (7) is now of interest. Computational times for several values of the equalizing parameter d are presented below, comparing our function axbyL and the Polynomial Toolbox for MATLAB. Roughly speaking, the equalizing parameter determines the number of previous samples that are used to estimate the output. At the CD sampling frequency of 44kHz, $d = 1000$ means for instance a delay of approximately 22ms, well acceptable in this audio setup.

d	`axbyc`	`axbycL`
100	$2.6s$	$0.23s$
300	$4.3s$	0.40
500	$17s$	$1.2s$
700	$44s$	$2.2s$
1000	$127s$	$6.3s$

Table 1. Computational times for various values of the equalizing parameter d. Polynomial Toolbox (`axbyc`) versus our routine `axbycL`.

8 Conclusions

An effective approach to numerical solution of polynomial Diophantine equations was presented in this paper. The proposed procedures are based on the relationship of Sylvester and block Sylvester matrices to general banded matrices and can be used both for scalar polynomials and

polynomial matrices. Suggested routines were implemented in the MATLAB environment using dedicated efficient solvers of the LAPACK library, supposedly the most powerful linear algebra package of today. The programs favorably compare to existing software tools for linear polynomial equations and can be employed successfully in demanding signal processing applications involving high degree polynomials.

The proposed banded-matrix approach seems to be a reasonable trade-off at this moment between reliable and available completely general linear equations solvers, used in the software tools for polynomial Diophantine equations at present, and the special algorithms tailored completely to the Sylvester structure that are under development now. The banded matrix approach leads definitely to significantly faster algorithms at almost no development and programming costs whereas the dedicated purely Sylvester procedures will presumably have to undertake a long path to prove themselves reliable and become widely available.

References

[1] Kučera V., *Analysis and Design of Discrete Linear Control Systems*, Academia Prague (1991).

[2] M. Vidyasagar, *Control Systems Synthesis: A Factorization Approach*, The MIT Press, London, 1987.

[3] V. Kucera, *The pole placement equation - a survey.* Kybernetika, Vol. 30, pp. 578-584, 1994.

[4] K. J. Hunt, *Polynomial Methods in Optimal Control and Filtering.* London:Peter Peregrinus, 1993. ISNB 0-86341-295-5.

[5] P. J. Antsaklis and Z.Gao, *Polynomial and Rational Matrix Interpolation: Theory and Control Applications*, Internetional Journal on Control. Vol 58, pp 349 - 404, 1993.

[6] D. Chen and C. Moler, *Symbolic Math Toolbox for Use with Matlab.* The MathWorks, Inc, 1993.

[7] S. Wolfram, *The Mathematica Book, Third Edition.* Cambridge University Press, 1996.

[8] D. Henrion, *Reliable Algorithms for Polynomial Matrices*, Ph.D. Thesis, Institute of Information Theory and Automation, Prague, 1998.

[9] Kwakernaak H., Šebek M., *PolyX Ltd. Web Site*, `http://www.polyx.com/`.

[10] L. Halmo and Z. Hurák, *PolPack++: Control-oriented library of numerical oriented algorithms for polynomial matrices in C/C++*, `http://sourceforge.net/projects/polpackplusplus/`.

[11] P. Kujan, M. Hromčík and M. Šebek, *New package for effective polynomial computation in Mathematica*, Proceedings of the 11th Mediterranean Conference on Control and Automation (MED03), Rhodes, Greece, June 18-20, 2003, ISBN 960-87706-0-2

[12] J. C. Zuniga and D. Henrion, *Block Toeplitz algorithms for polynomial matrix null-space computation*, LAAS-CNRS Research Report No. 04669, December 2004, on-line at `http://www.laas.fr/ henrion`.

[13] J. C. Zuniga and D. Henrion *A Toeplitz algorithm for polynomial J-spectral factorization*, Proceedings of the IFAC Symposium on System Structure and Control, Oaxaca, Mexico, December 8-10, 2004.

[14] LAPACK - Linear Algebra PACKage [online]. 2005 [cit 2005-01-05] http://www.netlib.org/lapack.

[15] BLAS - Basic Linear Algebra Subprogram [online]. 2005 [cit 2005-01-05] http://www.netlib.org/lapack.

[16] LINPACK [online]. 2005 [cit 2005-01-05] http://www.netlib.org/lapack.

[17] EISPACK [online]. 2005 [cit 2005-01-05] http://www.netlib.org/lapack.

[18] The MathWorks. Matlab [online]. 2005 [cit 2005-01-05] http://www.mathworks.com

[19] M. Sternad, M. Johansson and J. Rutstrom, *Inversion of Loudspeaker Dynamics by Polynomial LQ Feedforward Control*, Proceedings of the 3rd IFAC Symposium on Robust Control Design ROCOND 2000, Prague, CZ, June 21-23, 2000.