ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ
KATEDRA ŘÍDICÍ TECHNIKY

DIPLOMOVÁ PRÁCE

Algorithms for structured matrices with aplications in polynomial design methods

Praha, 2005 **Radek Frízel**

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb. , o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne _____ _____

podpis

Acknowledgements

I would like to thank to my advisor Martin Hromčík for his guidance, support and valuable advices.

Thanks to Petr Augusta, Ondřej Holub, Jakub Holý, Josef Malíř and Zdeněk Hurák for their comments and many interesting discussions.

I would also like to thank my parents and my family for their care and support in my life.

Abstrakt

Tato diplomová práce ukazuje možnost zrychlení stávajících postupů pro výpočet maticových rovnic na základě znalosti struktury matic. Zaměřuje se na řešení polynomiální Diofantické rovnice, často potřebné při analýze systému a návrhu regulačních obvodů a filtrů, pro které je rychlost jejího výpočtu podstatná. Byla implementována funkce pro prostředí Matlab, založená na Sylvestrově metodě, která dosahuje zrychlení při zachování přesnosti výpočtu. Funkce spolupracuje s knihovnou "Polynomial toolbox for Matlab" a využívá externího solveru LAPACKu, umožnujícího efektivnější výpočet matic se známou strukturou.

Abstract

This diploma thesis shows possibilities of acceleration of existing methods for computation of matrices equations and is based on the knowledge of the matrix structure. It focuses on solving the Diophantine equation, which is often required in analysis of control systems where computation speed plays an essential role. A function for MATLAB based on Sylvester method have been implemented, which accelerates the method while preserving the precision of solution. This function co-operates with the ”Polynomial Toolbox for Matlab” and uses an external solver LAPACK that has functions for matrices of known structure.

vložit originální zadáníííííííííííííííí !!!!!

Contents

Chapter 1

Introduction

The purpose of this thesis is to explore the possibilities of improving the algorithm which solves Diophantine equations based on the knowledge of the matrix structure. To profit from the special structure we decided to solve the polynomial Diophantine equation through the Sylvester method using an external solver. There are no solvers for the special structure of Sylvester matrix, so we chose the solver LAPACK, because it contains functions for band matrices that cover Sylvester case. We insert the Sylvester matrix in a general band matrix since the structure of Sylvester matrix is a special case of band matrices. Because of certain limitations of LAPACK we had to implement some wrapper routines.

1.1 Polynomial design methods

Speaking in broad terms, we can distinguish three main approaches to analysis and design of linear control systems.

- The classical frequency-domain methods have evolved from the analysis of frequency responses of linear dynamics systems. Their main formal mathematical tool is the theory of functions of complex variable, in particular the Laplace transform in case of continuous-time and the Z-transform for the discrete-time systems. Systems are described in terms of their transfer functions reflecting just the external input-output relations, which brings about some difficulties related to the internal stability of the closed loop and to the realization of the compensator. The used formalism also causes that the domain of classical methods is reduced to SISO time-invariant linear systems. But despite these limitations, the classical methods still remain very popular, namely in the community of practicing engineers, for their simplicity and

effectivity in many control problems encountered in industry.

The classical methods are suitable for computational processing and a lot of software tools supporting the design process are available.

- The drawbacks of the classical approach and the increasing complexity of systems to be controlled resulted in new methods of synthesis, usually called the state-space or modern approach. The methods rely upon the exact definition of the state that is systematically used both for the deeper analysis of the plant (the state provides the insight into the internal structure of the system) and for the synthesis of the compensator (the knowledge of the state is employed for compensation). The main formal tools are differential equations, vector spaces and matrix theory. The modern methods are applicable to much wider class of situations than the classical ones, e.g. to MIMO and time-varying systems. However, they have not become so popular, namely among the practicing engineers, for the necessity of finding the state-space model and for the need of state reconstruction in case it cannot be directly measured.

  From the numerical point of view, the state-space design methods for linear systems rely on numerical linear algebra which is a powerful tool. Since the 50's a lot of effort has been devoted to the development of accurate and numerically stable algorithms for linear algebra problems encountered in a large number of scientific computations.

- The origin of the polynomial or algebraic approach is dated to the early 70's. The polynomial matrices forming polynomial matrix fractions are introduced to handle MIMO cases. Systems are described by input-output relations, however the transfer functions are not regarded as functions of complex variable but as algebraic objects. The design procedure is then reduced to algebraic operations with rational and polynomial matrices, typically to solving algebraic Diophantine equations. This approach does not only enable us to resolve many existing control problems in 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.

  The algebraic methods are closely related to the Czech science. Among many others, let us remember prof. Vladimír Kučera who is well known as the pioneer of algebraic approach in the field of control theory [11, 12, 13].

1.2 Diophantine equations in control

We will not exaggerate too much if we say that almost all polynomial or algebraic designs lead to linear equations in the form

$$\boldsymbol{A}(s)\,\boldsymbol{X}(s) + \boldsymbol{B}(s)\,\boldsymbol{Y}(s) = \boldsymbol{C}(s), \tag{1.1}$$

where $\boldsymbol{A}(s)$, $\boldsymbol{B}(s)$ and $\boldsymbol{C}(s)$ are polynomials or polynomial matrices. These equations are called Diophantine polynomial equations.

We recall a few cases in the following.

- Pole placement

  Pole placement design is often used to stabilize an unstable plant or to achieve required tracking properties. This is done by replacing the original plant's poles with others that are for some reason more *desirable*, [10].

- Minimum variance control

  Let us consider a stochastic system where the plant output is distributed by an additive disturbance modelled as an output of a linear filter driven by white noise. Here the Diophantine equation is essential for finding the controller. The control objective is to minimize the variance of the output k steps ahead, given information up to the current time only.

  The standard approach to this problem is to use the system model to make prediction for the output k steps into the future and then set this prediction (or rather that part of it that can be influences by feedback) to zero [14].

- Predictive control

  Predictive control involves prediction for horizon longer then the time delay. Whenever there is a prediction, we can also expect a Diophantine equation to arise - in fact as the predictive cost function often involves a summation, there will be also more then one Diophantine equation. However, unlike the minimum variance control case, the solution of these equations will not normally appear explicitly in the controller [14, 20].

- Robust stabilization

  We shall now consider the design of controller for imprecisely known plants. Thus a nominal plant description is available, together with a description of the plant uncertainty, and the objective is to design a controller that stabilizes all plants lying

within the specified band of uncertainly. Such the controller is said to *robustly stabilize* the family of plants [20].

Diophantine equation plays important role also in other solutions of control problems like LQ, reference tracking, finite impulse response, non-linear systems etc[20, 21].

We briefly discuss existence of solution of Diophantine polynomial equation below and describe main methods to solve it: Interpolation, symbolic methods and most useful Sylvester method.

Diophantine equation in scalar case is solvable if and only if every common divisor of $\boldsymbol{a}(s)$ and $\boldsymbol{b}(s)$ is a divisor of $\boldsymbol{c}(s)$. Since the Diophantine equation is linear, any and all solutions pairs of 1.1 are given by

$$\boldsymbol{x}(s) = \boldsymbol{x}_0(s) + \boldsymbol{b}(s)\,\boldsymbol{q}(s), \boldsymbol{y}(s) = \boldsymbol{y}_0(s) - \boldsymbol{a}(s)\,\boldsymbol{q}(s), \tag{1.2}$$

where

$$\boldsymbol{q}(s) = \boldsymbol{q}_0 + \boldsymbol{q}_1\,\boldsymbol{s} + \boldsymbol{q}_2\,\boldsymbol{s}^2..$$

is an arbitrary polynomial such that $\boldsymbol{x}_0(s) + \boldsymbol{b}(s)\,\boldsymbol{q}(s)$ is non-zero.

Polynomial matrix Diophantine equations can be handled alike. In general, if $\boldsymbol{A}(s)$, $\boldsymbol{B}(s)$ and $\boldsymbol{C}(s)$ are matrices of dimensions *m x p*, *m x k* and *m x q*, respectively, then equation 1.1 has a solution $\{\, \boldsymbol{X}_0(s), \boldsymbol{Y}_0(s) \,\}$ if and only if any common divisor of $\boldsymbol{A}(s)$ and $\boldsymbol{B}(s)$ is a left divisor of $\boldsymbol{C}$. Any and all solutions pairs of Diophantine equation 1.1 in matrix case are given by

$$\boldsymbol{X}(s) = \boldsymbol{X}_0(s) + \boldsymbol{B}(s)\,\boldsymbol{Q}(s), \boldsymbol{Y}(s) = \boldsymbol{Y}_0(s) - \boldsymbol{A}(s)\,\boldsymbol{Q}(s), \tag{1.3}$$

where

$$\boldsymbol{Q}(s) = \boldsymbol{Q}_0 + \boldsymbol{Q}_1\,\boldsymbol{s} + \boldsymbol{Q}_2\,\boldsymbol{s}^2..$$

is an arbitrary polynomial matrix such that $\boldsymbol{X}_0(s) + \boldsymbol{B}(s)\,\boldsymbol{Q}(s)$ is non-zero.
Because matrices multiplication isn't commutative, there is also equation

$$\boldsymbol{X}(s)\,\boldsymbol{A}(s) + \boldsymbol{Y}(s)\,\boldsymbol{B}(s) = \boldsymbol{C}(s), \tag{1.4}$$

where $\boldsymbol{A}(s)$, $\boldsymbol{B}(s)$ and $\boldsymbol{C}(s)$ are polynomial matrices. With this equation can be handled alike as with the Diophantine equation.

1.3 Numerical algorithms for Diophantine equations

The existing methods for solving of Diophantine equations are:

- Interpolation
- Symbolic procedures
- Sylvester matrix method

1.3.1 Interpolation

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

The basic idea is as follows: The input polynomial matrices are evaluated at suitably chosen point at first. In this way, an equivalent representation of the problem in terms of constant matrices 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.

The solution to the most parts of control problems involving polynomial matrices including Diophantine equations via this approach have been given in the survey paper [15].

The problematic points of the methods are the estimation of the number of interpolation points and their placement.

The choice of the interpolation points appears crucial. The numerical reliability of the particular interpolation algorithm strongly depends on the location of the points. Usually the points are chosen either equidistantly spaced on the real or imaginary axis (see for instance [19]) or random in some way [17]. In the sequel we will give the novel results showing the remarkable benefit of the use of Fourier points in this context.

1.3.2 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 [18], MATHEMATICA [16]). 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 more, the execution time increased usually exponentially with the size of task. The efficiency of the methods also depend on the particular data - for instance they are much faster for matrices 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 alghorithm. These limitations make this approach practically unusable for the purposes of control systems design.

On the positive side, the methods return precise results for small size problems when the execution time and memory requirements are acceptable.

1.3.3 Sylvester matrix method

Many computations with polynomial matrices can be expressed in terms of related Sylvester matrix. This interpretation is straightforward and leads to the set of constant linear equations the numerical methods of which are well understood. On the other hand the size of the related Sylvester matrices are usually rather high and the method when implemented require a large amount of memory. For all that the Sylvester method is most useful from all remarked methods.

The first step is to transfer Diophantine equation 1.1 to a special form of matrix polynomial equation.

$$\widehat{\boldsymbol{A}}(s)\,\widehat{\boldsymbol{X}}(s) = \widehat{\boldsymbol{B}}(s), \tag{1.5}$$

A parameterisation of solution is

$$\widehat{\boldsymbol{A}}(s) = [\boldsymbol{A}(s)\boldsymbol{B}(s)], \widehat{\boldsymbol{X}}(s) = \begin{bmatrix} \mathbf{X}(\mathrm{s}) \\ \mathbf{Y}(\mathrm{s}) \end{bmatrix}, \widehat{\boldsymbol{B}}(s) = \boldsymbol{C}(s), \tag{1.6}$$

Next we find solution of special form of matrix polynomial equation 1.5 where $\widehat{\boldsymbol{A}}(s)$ is a given *m x n* polynomial matrix, $\widehat{\boldsymbol{B}}(s)$ is a given *m x p* polynomial matrix and $\widehat{\boldsymbol{X}}(s)$ is an *n x p* polynomial matrix to be found. Write the polynomial matrices in term of increasing powers of the indeterminate s, i.e.

$$\begin{aligned} \boldsymbol{A}(s) &= \boldsymbol{A}_0 + s\,\boldsymbol{A}_1 + ... + s^{d_A}\,\boldsymbol{A}_{d_A} \\ \boldsymbol{B}(s) &= \boldsymbol{B}_0 + s\,\boldsymbol{B}_1 + ... + s^{d_B}\,\boldsymbol{B}_{d_B} \\ \boldsymbol{X}(s) &= \boldsymbol{X}_0 + s\,\boldsymbol{X}_1 + ... + s^{d_X}\,\boldsymbol{X}_{d_X} \end{aligned}$$

where $d_A = degA(s)$, $d_B = degB(s)$ and $d_X = degX(s)$. From the equation 1.5 flows the unequation $d_A + d_X \leq d_B$. The equality does not hold, if some leading matrix

coefficients in $\widehat{\boldsymbol{A}}(s)$ or $\widehat{\boldsymbol{B}}(s)$ is zero. Thanks to the equality of the powers of s in the above matrix polynomial equation, we can build an equivalent linear system of equations

$$\underbrace{\begin{bmatrix} A_0 & & & 0 \\ A_1 & A_0 & & \\ \vdots & A_1 & \ddots & \\ A_{d_A} & \vdots & & A_0 \\ & A_{d_A} & & A_1 \\ & & \ddots & \vdots \\ 0 & & & A_{d_A} \end{bmatrix}}_{\langle A \rangle_{d_X}} \underbrace{\begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{d_X} \end{bmatrix}}_{\overline{X}} = \underbrace{\begin{bmatrix} B_0 \\ B_1 \\ \vdots \\ B_{d_B} \end{bmatrix}}_{\overline{B}}, \tag{1.7}$$

Matrix $\langle A_{d_X} \rangle$ is referred to as the Sylvester matrix of $\widehat{\boldsymbol{A}}(s)$ of order d_X. It has $(d_A + d_X + 1)\, m$ rows and $(d_X + 1)$ *n* columns. Since matrix polynomial equation 1.5 and linear system 1.7 are equivalent, solving 1.7 for a constant solution $\overline{X}$ amounts to solving 1.5 for a polynomial solution $\widehat{\boldsymbol{X}}(s)$.

The problematic point in this approach is clearly the estimation of d_X, the degree of the solution $\widehat{\boldsymbol{X}}(s)$. This is a key parameter influencing the size of the Sylvester matrix, thus the number of equations and unknowns in the equivalent linear system of equation 1.7. We shall see that, in some cases, an upper bound on d_X can be found. In other cases, some assumptions on matrices $\widehat{\boldsymbol{A}}(s)$ and $\widehat{\boldsymbol{B}}(s)$ will guarantee the existence of solution $\widehat{\boldsymbol{X}}(s)$of a given degree.

1.4 Structure of Sylvester and block Sylvester matrices

Before finishing this introduction we would like to describe different structure of Sylvester and block Sylvester matrix.

When we build Sylvester matrix from polynomial, i.e.

$$\boldsymbol{A}(s) = \boldsymbol{a}_0 + s\,\boldsymbol{a}_1 + ... + s^{d_A}\,\boldsymbol{a}_{d_A}$$

where a_i is a scalar and $d_A = degA(s)$, as is shown in section 1.3.3, we get the Sylvester

matrix like

$$\begin{bmatrix} a_0 & & & 0 \\ a_1 & a_0 & & \\ \vdots & a_1 & \ddots & \\ a_{d_A} & \vdots & & a_0 \\ & a_{d_A} & & a_1 \\ & & \ddots & \vdots \\ 0 & & & a_{d_A} \end{bmatrix}. \tag{1.8}$$

On the other hand when we build Sylvester matrix from polynomial matrices

$$\boldsymbol{A}(s) = \boldsymbol{A}_0 + s\,\boldsymbol{A}_1 + ... + s^{d_A}\,\boldsymbol{A}_{d_A}$$

where $\boldsymbol{A}_i$ is *m x n* matrix

$$\boldsymbol{A}_i = \begin{bmatrix} a^i_{1,1} & a^i_{1,2} & \dots & a^i_{1,n} \\ a^i_{2,1} & a^i_{2,2} & \dots & a^i_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a^i_{m,1} & a^i_{m,2} & \dots & a^i_{m,n} \end{bmatrix}$$

and $d_A = degA(s)$, we get the block Sylvester matrix. For example see the block Sylvester matrix build from *3 x 3* matrix of polynomial degree d_A in the equation 1.9.

The difference between structure of Sylvester matrix and block Sylvester matrix is obvious from the matrix 1.8 and the matrix 1.9.

The Sylvester matrix is lower triangular band matrix. It has values only under or on the diagonal. The block Sylvester matrix is special band matrix with the number of superdiagonal same as the number of rows in the coefficient matrices of matrix from which we have the Sylvester matrix built.

$$\begin{bmatrix} \begin{bmatrix} a^0_{1,1} & a^0_{1,2} & a^0_{1,3} \\ a^0_{2,1} & a^0_{2,2} & a^0_{2,3} \\ a^0_{3,1} & a^0_{3,2} & a^0_{3,3} \end{bmatrix} & & & 0 \\ \begin{bmatrix} a^1_{1,1} & a^1_{1,2} & a^1_{1,3} \\ a^1_{2,1} & a^1_{2,2} & a^1_{2,3} \\ a^1_{3,1} & a^1_{3,2} & a^1_{3,3} \end{bmatrix} & \begin{bmatrix} a^0_{1,1} & a^0_{1,2} & a^0_{1,3} \\ a^0_{2,1} & a^0_{2,2} & a^0_{2,3} \\ a^0_{3,1} & a^0_{3,2} & a^0_{3,3} \end{bmatrix} & & \\ \vdots & \begin{bmatrix} a^1_{1,1} & a^1_{1,2} & a^1_{1,3} \\ a^1_{2,1} & a^1_{2,2} & a^1_{2,3} \\ a^1_{3,1} & a^1_{3,2} & a^1_{3,3} \end{bmatrix} & \ddots & \\ \begin{bmatrix} a^{d_A}_{1,1} & a^{d_A}_{1,2} & a^{d_A}_{1,3} \\ a^{d_A}_{2,1} & a^{d_A}_{2,2} & a^{d_A}_{2,3} \\ a^{d_A}_{3,1} & a^{d_A}_{3,2} & a^{d_A}_{3,3} \end{bmatrix} & \vdots & & \begin{bmatrix} a^0_{1,1} & a^0_{1,2} & a^0_{1,3} \\ a^0_{2,1} & a^0_{2,2} & a^0_{2,3} \\ a^0_{3,1} & a^0_{3,2} & a^0_{3,3} \end{bmatrix} \\ & \begin{bmatrix} a^{d_A}_{1,1} & a^{d_A}_{1,2} & a^{d_A}_{1,3} \\ a^{d_A}_{2,1} & a^{d_A}_{2,2} & a^{d_A}_{2,3} \\ a^{d_A}_{3,1} & a^{d_A}_{3,2} & a^{d_A}_{3,3} \end{bmatrix} & & \begin{bmatrix} a^1_{1,1} & a^1_{1,2} & a^1_{1,3} \\ a^1_{2,1} & a^1_{2,2} & a^1_{2,3} \\ a^1_{3,1} & a^1_{3,2} & a^1_{3,3} \end{bmatrix} \\ & & \ddots & \vdots \\ 0 & & & \begin{bmatrix} a^{d_A}_{1,1} & a^{d_A}_{1,2} & a^{d_A}_{1,3} \\ a^{d_A}_{2,1} & a^{d_A}_{2,2} & a^{d_A}_{2,3} \\ a^{d_A}_{3,1} & a^{d_A}_{3,2} & a^{d_A}_{3,3} \end{bmatrix} \end{bmatrix}. \quad (1.9)$$

Chapter 2

Objectives

The objectives of this work is to explore the possibilities of using an external solver that profits from a knowledge of special structure of matrices to improve the existing methods for polynomial and structured matrices.

We decided to test our idea on the solution of Diophantine equation through the Sylvester method. The reason of our choice was that the Diophantine equations are very often solved in control theory and they are indispensable part of control synthesis.

We didn't want to design new algorithms because we wanted to accelerate calculations of polynomial and structured matrices in practice control design in short time and without big cost. Design of new algorithms is work for a long time and mathematical hard.

Chapter 3

Implementation

In chapter Introduction we made an overview of methods for solving polynomial Diophantine equation. We also explained Sylvester approach and structure of Sylvester matrix. In this chapter we present our implementation of Sylvester method to solve Diophantine equation using external banded matrix solver.

3.1 Selection of software platform

As a software framework for implementation and for testing of improving the existing algorithms for Sylvester method of solving Diophantine equation based on knowledge of the matrix structure the MATLAB and the Polynomial Toolbox for MATLAB were taken. For computation with band matrices we took the solver LAPACK. There were several reasons for this decision, indicated below.

All these have been implemented and tested under Microsoft Windows. According to the documentation there might be slight differences under other operation system.

3.1.1 Polynomial Toolbox

The Polynomial Toolbox (PT), provided by the Polyx, Ltd company [8], offers objects and functions for easy manipulation with polynomials and polynomial matrices. It offers standard functions for handling polynomials such as addition, determinant, computing divisor, solving algebraic equation, etc. The PT is moreover focused on control analysis and synthesis problems. Solving of Diophantine equation is also implemented in the PT and it is the reference function for our modified algorithm and benchmarks.

A simple example of work with PT follows. Suppose three polynomial matrices A, B and C

$>> \boldsymbol{A} = [1+s, 1; 1-s, 1]$,$\boldsymbol{B} = [1, 1-s; 1, 1+s]$,$\boldsymbol{C} = [1+s, 1+s; 1+s, 1+s]$

$\boldsymbol{A} =$

$$\begin{matrix} 1+s & 1 \\ 1-s & 1 \end{matrix}$$

$\boldsymbol{B} =$

$$\begin{matrix} 1 & 1-s \\ 1 & 1+s \end{matrix}$$

$\boldsymbol{C} =$

$$\begin{matrix} 1+s & 1+s \\ 1+s & 1+s \end{matrix}$$

thus the matrices $\boldsymbol{A}$,$\boldsymbol{B}$ and $\boldsymbol{C}$ are set we use PT's function for find solution $[\boldsymbol{X}\ \boldsymbol{Y}]$ of Diophantine equation $\boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{Y} = \boldsymbol{C}$

$>> [\boldsymbol{X}, \boldsymbol{Y}] = axbyc(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$

$\boldsymbol{X} =$

$$\begin{matrix} 0.25+0.25s & 0.25+0.25s \\ 0.25+0.25s & 0.25+0.25s \end{matrix}$$

$\boldsymbol{Y} =$

$$\begin{matrix} 0.25+0.25s & 0.25+0.25s \\ 0.25+0.25s & 0.25+0.25s \end{matrix}$$

it is easy to test the solution

$>> \boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{Y} - \boldsymbol{C}$

$$\begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$$

if we want to know a Sylvester matrix build from matrix $\boldsymbol{A}$

$>> \boldsymbol{S}A = sylv(\boldsymbol{A}\text{,2,'}col\text{'})$

$\boldsymbol{S}A =$

$$\begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{matrix}$$

3.1.2 LAPACK

LAPACK is written in Fortran 77 and provides routines for solving systems of simultaneous 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 provided, as are related computations such as reordering of the Schur factorisations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision. LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS) [4].

The original goal of the LAPACK project was to make the widely used EISPACK [6] and LINPACK [5] 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 routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). While LINPACK and EISPACK are based on the vector operation kernels of the Level 1 BLAS, LAPACK was designed at the outset to exploit the Level 3 BLAS – a set of specifications for Fortran subprograms that do various types of matrix multiplication and the solution of triangular systems with multiple right-hand sides. Because of the coarse granularity of the Level 3 BLAS operations, their use promotes high efficiency on many high-performance computers.

LAPACK is a freely available software package provided on the World Wide Web via netlib, anonymous ftp, and http access [3]. Thus it can be and has been included in commercial packages (MATLAB).

3.1.3 MATLAB and LAPACK

For using LAPACK's functions in MATLAB, the company MathWorks provides an external interface, MEX-files. These files can be written in programming language C or Fortran. It is a gate between Matlab and functions from LAPACK library. Calling the well-known LAPACK library as external solver is well described for example in the Matlab Help or in [7].

We explain creating of MEX-file on example of calling LAPACK function, which compute LU factorisation of band rectangular matrix, dbgtrf.f. The example is in language C.

MEX-file has set structure. It contains function *mexFunction*, which is interface between MATLAB and LAPACK functions. This mex function has four parameters: number of input parameters, array of pointers to input parameters, numbers of output parameters and array of pointers to output parameters. To retrieve the values of input parameters and store them into local variables mexFunction uses appropriate MATLAB functions. When it's done we call the LAPACK function. The pointers to output are stored to array of the mexFuction and the local arrays are destroyed.

Code of MEX file with explanatory comment:

```
#include "mex.h"
// interface function
void mexFunction(int nlhs, mxArray *plhs[],
        int nrhs, const mxArray *prhs[])
{
```

```
//declare variable
double *AB, *temp, *temp1, *temp2, zero=0.0;
int *IPIV;
int LDAB, M, N, Info, k, KL, KU, min, MB, NB;
char msg[100];
//write message to MATLAB workspace
 mexPrintf("\n Start mex interface \n");
//Test of number input/otput parameter
 if (nrhs != 5)
   {
        mexErrMsgTxt("Five input arguments required.");
   } else if (nlhs > 2)
   {
        mexErrMsgTxt("Too many output arguments.");
   };

//read of input parameters
//size of first inpu parameter, matrix AB
 MB = mxGetM(prhs[0]);
 NB = mxGetN(prhs[0]);
//store first input parameter into local variable AB
 AB = mxCalloc (MB*NB, sizeof(double));
 temp = mxGetPr(prhs[0]);
 for (k=0;k<NB*MB;k++)
      {
      AB[k] = temp[k];
      };
//read of last four parameters into local variables
//(number of subdiagonal, supperdiagonal, size of matrix A)
if (!mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
       mxGetN(prhs[1])*mxGetM(prhs[1]) != 1) {
    mexErrMsgTxt("Input KL must be a scalar.");
  }
  KL = mxGetScalar(prhs[1]);
if (!mxIsDouble(prhs[2]) || mxIsComplex(prhs[2]) ||
       mxGetN(prhs[2])*mxGetM(prhs[2]) != 1) {
```

```
    mexErrMsgTxt("Input KU must be a scalar.");
  }
  KU = mxGetScalar(prhs[2]);
if (!mxIsDouble(prhs[3]) || mxIsComplex(prhs[3]) ||
      mxGetN(prhs[3])*mxGetM(prhs[3]) != 1) {
    mexErrMsgTxt("Input M must be a scalar.");
  }
  M = mxGetScalar(prhs[3]);
if (!mxIsDouble(prhs[4]) || mxIsComplex(prhs[4]) ||
      mxGetN(prhs[4])*mxGetM(prhs[4]) != 1) {
    mexErrMsgTxt("Input N must be a scalar.");
  }
  N = mxGetScalar(prhs[4]);
if (M>N)
    min = N;
else
    min = M;

// set necessary parameters for calling LAPACK function
//Leading dimension of matrix AB
LDAB=MB;
IPIV = mxCalloc (min, sizeof(int));

//Call LAPACK function
dgbtrf(&M,&N, &KL,&KU,AB,&LDAB,IPIV,&Info);

if (Info < zero)
  {
    sprintf(msg,"Input %d to DGBTRF had an illegal value",-Info);
    mexErrMsgTxt(msg);
  };

//store output parameters
//factorizated matrix
 plhs[0] = mxCreateDoubleMatrix(MB,NB,mxREAL);
 temp1 = mxCalloc(MB*NB, sizeof(double));
```

```
 for (k = 0; k < MB*NB; k++)
   {
      temp1[k] = AB[k];
   }
 mxSetPr(plhs[0],temp1);
// IPIV array, array of permutations
 plhs[1] = mxCreateDoubleMatrix(1,min,mxREAL);
 temp2 = mxCalloc(min, sizeof(double));
  for (k = 0; k < min; k++)
  {
     temp2[k] = (double)IPIV[k];
  }
 mxSetPr(plhs[1], temp2);

// free local variables
  mxFree(AB);
  mxFree(IPIV);
return;
}
```

To compile and link the MEX-file at the MATLAB prompt, type:
mex mex_file_name.c {MATLAB_home}$\backslash extern\backslash lib\backslash win32\backslash lcc\backslash libmwlapack.lib$.

For correct recalling of parameters of MATLAB's mex function see *help mex* in MATLAB.

3.2 Implementated functions

In this section we discuss integration of external solver in the calculation of Diophantine equation and taking advantage of the knowledge of the structure of resolved matrices.

Our function finds a particular solution $[\boldsymbol{X}_0,\boldsymbol{Y}_0]$ of the polynomial and matrix polynomial Diophantine equation $\boldsymbol{AX} + \boldsymbol{BY} = \boldsymbol{C}$. If no polynomial solution exists then all the entries of $[\boldsymbol{X}_0, \boldsymbol{Y}_0]$ are set equal to NaN. The solution is finding through Sylvester method using the LAPACK and the minimum degree is finding through Binary search.

Interfaces of the LAPACK's functions that we need to call when solving the Diophantine equations must be compiled in advance. By interfaces we mean mex-files whose creation is described above - see 3.1.3. The mex-files are compiled into dynamical li-

braries and MATLAB works with them in the same manner as with standard MATLAB's m-files.

In our function we also use auxiliary MATALAB function *STORAGE2* that stores band matrix in required form for LAPACK. See below for further details. An *m x n* band matrix with *kl* subdiagonals and *ku* superdiagonals may be stored compactly in a two-dimensional array with $2 * kl + ku + 1$ rows and *n* columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array as follows:

$$\boldsymbol{A}B\,(kl + ku + 1 + i - j, j) = \boldsymbol{A}\,(i, j)\ for\ max(1, j - ku) <= i <= min(m, j + kl)$$

This storage scheme should be used in practice only if kl, $ku << min(m, n)$.
See example of storage:
band matrix $\boldsymbol{A}$

$$A = \begin{bmatrix} a11 & a12 & 0 & 0 & 0 & 0 & 0 \\ a21 & a22 & a23 & 0 & 0 & 0 & 0 \\ a31 & a32 & a33 & a34 & 0 & 0 & 0 \\ 0 & a42 & a43 & a44 & a45 & 0 & 0 \\ 0 & 0 & a53 & a54 & a55 & a56 & 0 \\ 0 & 0 & 0 & a64 & a65 & a66 & a67 \end{bmatrix}$$

is stored into $2 * kl + ku + 1$ matrix $\boldsymbol{A}B$

$$AB = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a12 & a23 & a34 & a45 & a56 & a67 \\ a11 & a22 & a33 & a44 & a55 & a66 & 0 \\ a21 & a32 & a43 & a54 & a65 & 0 & 0 \\ a31 & a42 & a53 & a64 & 0 & 0 & 0 \end{bmatrix}$$

First *kl* rows are used in LAPACK function that computes LU factorisaton of band matrix.

Now we describe our function for types of Diophantine equations.

3.2.1 Scalar case

- Description

  First part of our function *axbycL* finds solution of the scalar case of polynomial Diophantine equation

  $$\boldsymbol{A}(s)\,\boldsymbol{X}(s) + \boldsymbol{B}(s)\,\boldsymbol{Y}(s) = \boldsymbol{C}(s), \tag{3.1}$$

  where $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are polynomials.

- Syntax

  $[\boldsymbol{X}, \boldsymbol{Y}] = axbycL(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$

- Method

  First part of function *axbycL* finds solution $[\boldsymbol{X}, \boldsymbol{Y}]$ of the polynomial Diophantine equation 3.1 through Sylvester method using external solver Lapack.

  First the degree of solution is set according to
  $deg\boldsymbol{X} = deg\boldsymbol{B}$ - 1,
  $deg\boldsymbol{Y} = deg\boldsymbol{A}$ - 1 pro $deg\boldsymbol{A} + deg\boldsymbol{B} > deg\boldsymbol{C}$,
  $deg\boldsymbol{Y} = deg\boldsymbol{C}$ - $deg\boldsymbol{B}$ pro $deg\boldsymbol{A} + deg\boldsymbol{B} \leq deg\boldsymbol{C}$.

  Next the Sylvester matrices $\boldsymbol{S}A$ and $\boldsymbol{S}B$ from polynomial $\boldsymbol{A}$ and polynomial $\boldsymbol{B}$ are created . The orders of matrices are determined by degrees of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively.

  Then a constant matrix equation $\boldsymbol{S}\,\boldsymbol{X}Y = \boldsymbol{C}_c$ is composed and computed by LAPACK function. Matrix $\boldsymbol{S}$ is double Sylvester matrix joined from the Sylvester matrices $\boldsymbol{S}A$ and $\boldsymbol{S}B$. Vector $\boldsymbol{C}_c$ is composed from coefficients of polynomial $\boldsymbol{C}$. The vector $\boldsymbol{X}Y$ contains coefficients of finding solution of Diophantine equation.

  $$\boldsymbol{X}Y = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{degX} \\ y_0 \\ y_1 \\ \vdots \\ y_{degy} \end{bmatrix} \tag{3.2}$$

LAPACK doesn't contain functions for compute linear equations with block Sylvester matrix 1.4 so we used function for compute band matrix because structure of block Sylvester matrix is close to structure of band matrix see below.

$$\text{Sylvester matrix} = \begin{bmatrix} a1 & 0 & 0 & 0 & b1 & 0 & 0 \\ a2 & a1 & 0 & 0 & b2 & b1 & 0 \\ a3 & a2 & a1 & 0 & b3 & b2 & b1 \\ 0 & a3 & a2 & a1 & b4 & b3 & b2 \\ 0 & 0 & a3 & a2 & 0 & b4 & b3 \\ 0 & 0 & 0 & a3 & 0 & 0 & b4 \end{bmatrix}$$

$$\text{band matrix} = \begin{bmatrix} xx & xx & xx & xx & xx & 0 & 0 \\ xx & xx & xx & xx & xx & xx & 0 \\ xx & xx & xx & xx & xx & xx & xx \\ 0 & xx & xx & xx & xx & xx & xx \\ 0 & 0 & xx & xx & xx & xx & xx \\ 0 & 0 & 0 & xx & xx & xx & xx \end{bmatrix}$$

- Example

  Suppose three polynomials

  $>> A = [1+2*s+s^2+2*s^3], B = [s+s^2+3*s^3], C = [4+5*s+s^2+5*s^3+s^4]$

  $\boldsymbol{A} =$

  $$1 + 2s + s^2 + 2s^3$$

  $\boldsymbol{B} =$

  $$s + s^2 + 3s^3$$

  $\boldsymbol{C} =$

  $$4 + 5s + s^2 + 5s^3 + s^4$$

thus the polynomials $\boldsymbol{A}$,$\boldsymbol{B}$ and $\boldsymbol{C}$ are set we use our function *axbycL* for find solution $[\boldsymbol{X}\ \boldsymbol{Y}]$ of Diophantine equation $\boldsymbol{A}\,\boldsymbol{X} + \boldsymbol{B}\,\boldsymbol{Y} = \boldsymbol{C}$

$>> [\boldsymbol{X}, \boldsymbol{Y}] = axbycL(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$

$\boldsymbol{X} =$

$$4 - 2s + 0.36s^2$$

$\boldsymbol{Y} =$

$$-1 + 1.6s - 0.24s^2$$

3.2.2 Matrix case

- Description

  Second part of function *axbycL* finds a particular solution $[\boldsymbol{X}_0, \boldsymbol{Y}_0]$ of the matrix case of polynomial Diophantine equation

  $$\boldsymbol{A}(s)\,\boldsymbol{X}(s) + \boldsymbol{B}(s)\,\boldsymbol{Y}(s) = \boldsymbol{C}(s), \tag{3.3}$$

  where $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are polynomial matrices.

- Syntax

  $[\boldsymbol{X}, \boldsymbol{Y}] = axbycL(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$

  $[\boldsymbol{X}, \boldsymbol{Y}] = axbycL(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, DEGREE)$

  If *DEGREE* is not specified then a solution of minimum overall degree is computed. Otherwise function seeks a solution $[\boldsymbol{X}_0, \boldsymbol{Y}_0]$ of degree *DEGREE*.

- Method

  The process of our calculation of matrix Diophantine equation start with finding of upper and lower bound of degree of the solution. In scalar case, see 4.1, the degree

of the solution is exactly set. In the matrix case we can only found the bounds of the solution.

Minimum degree of solution is finding through binary search. Binary search is search of sorted array, in our case all possibilities degrees of the solution, by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key, degree in our case, is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

In the loop of the binary search we are finding the solution of the Diophantine equation using the LAPACK. The solution has degree destined by binary search.

The finding of the solution for test degree in the loop of binary search combines creation of constant linear equation with Sylvester matrix

$$\boldsymbol{D}\,\boldsymbol{X} = \boldsymbol{C}_c, \tag{3.4}$$

where $\boldsymbol{D}$ is the Sylvester matrix created from the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, matrix $\boldsymbol{C}_c$ is composed from coefficients of polynomial matrix $\boldsymbol{C}$ and the matrix $\boldsymbol{X}Y$ contains coefficients of finding solution of Diophantine equation, and finding of a solution of this equation. On base of size of Sylvester matrix is called one LAPACK's function or our routine with series of LAPACK's functions.

If the Sylvester matrix is square, not very frequent, we call same LAPACK's function as in the scalar case of Diophantine equation. This function solves constant linear equation $\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B}$, where $\boldsymbol{A}$ is square band matrix.

If the Sylvester matrix has more rows then columns, also not very frequent, we proceed from fact that if $\boldsymbol{A}$ is an *m x n* tall matrix where $m > n$, then $\boldsymbol{A}\backslash\boldsymbol{B}$ is the same as $(\boldsymbol{A}'\,\boldsymbol{A})\backslash(\boldsymbol{A}'\,\boldsymbol{B})$. Through this method we create from the tall band matrix a square band matrix and then we call also LAPACK's functions for solving constant linear equation $\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B}$, where $\boldsymbol{A}$ is square band matrix.

If Sylvester matrix has more columns then rows, we call our interface file with wrapper routines and LAPACK's functions. The process of finding solution of linear equation in mex file is as follow. We call LAPACK function for LU factorisation of band matrix. We get lower triangular matrix $\boldsymbol{L}$ and upper trapezoidal triangular matrix $\boldsymbol{U}$.

$$\boldsymbol{D} = \boldsymbol{L}\,\boldsymbol{U} \tag{3.5}$$

We multiply equation from left side with matrix $\boldsymbol{L}^{-1}$ and we get equation

$$\boldsymbol{U}\,\boldsymbol{X} = \boldsymbol{L}^{-1}\,\boldsymbol{B}. \tag{3.6}$$

Then we call LAPACK function for decomposition trapezoidal matrix $\boldsymbol{U}$ to the upper triangular form by means of orthogonal transformations. The *m x n* upper trapezoidal matrix $\boldsymbol{U}$ is factored as

$$\boldsymbol{U} = (\boldsymbol{R}\boldsymbol{0})\,\boldsymbol{Z}, \tag{3.7}$$

where $\boldsymbol{Z}$ is an *n x n* orthogonal matrix and $\boldsymbol{R}$ is an *m x m* upper triangular matrix. After this factorisation we call LAPACK's function that solves constant linear equation $\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B}$, where $\boldsymbol{A}$ is upper triangular square matrix and LAPACK's function that solves constant linear equation $\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B}$, where $\boldsymbol{A}$ is square orthogonal matrix.

If an error happens in LAPACK's function during the calculation, LAPACK does not signal it in any way and return an invalid result. Due to this we have to test correctness of solution.

- Example

  Suppose three polynomial matrices

  $>> \boldsymbol{A} = [1+s+s^2, 1+2*s+s^2, 1+s+2*s^2; 1+s^2, 1+3*s+s^2, 1+s; 2+s+s^2, 2+2*s^2, 1+s+s^2]$

  $\boldsymbol{A} =$

  $$\begin{matrix} 1+s+s^2 & 1+2s+s^2 & 1+s+2s^2 \\ 1+s^2 & 1+3s+s^2 & 1+s \\ 2+s+s^2 & 2+2s^2 & 1+s+s^2 \end{matrix}$$

  $>> \boldsymbol{B} = [3+3*s+s^2, 1+5*s+2*s^2, 2+8*s; 1+s+s^2, 1+3*s+2*s^2, 1+s+s^2; 1+s+s^2, s+2*s^2, 1+s+s^2]$

  $\boldsymbol{B} =$

  $$\begin{matrix} 3+3s+s^2 & 1+5s+2s^2 & 2+8s \\ 1+s+s^2 & 1+3s+2s^2 & 1+s+s^2 \\ 1+s+s^2 & s+2s^2 & 1+s+s^2 \end{matrix}$$

$>> \boldsymbol{C} = [5 + 5 * s + 5 * s^2, 1 + s + s^2, 2 + s + 6 * s^2; 1 + 4 + s + s^2, 3 + 4 * s + 5 * s^2, s + 3 * s^2; s^2, 1 + s + 2 * s^2, 3 + s + s^2]$

$\boldsymbol{C} =$

$$\begin{matrix} 5 + 5s + 5s^2 & 1 + s + s^2 & 2 + s + 6s^2 \\ 5 + s + s^2 & 3 + 4s + 5s^2 & s + 3s^2 \\ s^2 & 1 + s + 2s^2 & 3 + s + s^2 \end{matrix}$$

thus the polynomial matrices $\boldsymbol{A}$,$\boldsymbol{B}$ and $\boldsymbol{C}$ are set we use our function *axbycL* for find solution $[\boldsymbol{X}\ \boldsymbol{Y}]$ of Diophantine equation $\boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{Y} = \boldsymbol{C}$

$>> [\boldsymbol{X}, \boldsymbol{Y}] = axbycL(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$

$\boldsymbol{X} =$

$$\begin{matrix} -2 + 4.9s & 0.31 - 2.1s & 1.4 - 8s \\ -1.9 - 1.3s & -0.52 + s & 0.63 + 1.7s \\ 4.8 + 1.3s & 3.1 - s & 0.96 - 1.7s \end{matrix}$$

$\boldsymbol{Y} =$

$$\begin{matrix} -2.9 - 18s & -0.26 + 2.7s & 4 + 19s \\ 1.1 + 5.9s & 1.8 + 0.2s & -0.97 - 4.8s \\ 5.9 + 2.6s & -1.5 - 2.1s & -6 - 3.5s \end{matrix}$$

if we can find solution of degree 2, write command

$>> [\boldsymbol{X}, \boldsymbol{Y}] = axbycL(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, 2)$

$\boldsymbol{X} =$

$$\begin{matrix} -2.4 - 0.67s + 0.75s^2 & 0.45 - 0.65s - 0.037s^2 & 2 - 0.65s - 0.66s^2 \\ -1.9 + 0.3s - 0.29s^2 & -0.46 + 0.73s - 0.021s^2 & 0.94 + 0.17s + 0.063s^2 \\ 8 - 2.7s + 0.29s^2 & 2.3 - 0.61s + 0.021s^2 & -3.4 + 0.96s - 0.063s^2 \end{matrix}$$

$$\boldsymbol{Y} =$$

$$\begin{matrix} -0.73 - 2s - 0.33s^2 & -0.76 - 0.95s - 0.13s^2 & 1.4 + 0.47s + 0.43s^2 \\ 0.65 + 2.7s - 0.36s^2 & 2 + 0.92s + 0.074s^2 & -0.095 - 1.2s + 0.14s^2 \\ 1.5 - 1.5s + 0.59s^2 & -0.49 - s + 0.043s^2 & -0.88 + 1.5s - 0.13s^2 \end{matrix}$$

Chapter 4

Numerical experiments

We compare Polynomial toolbox and our function for finding the solution of polynomial Diophantine equation. We are interested in the result precision and the computing time. We presume our functions are faster then function from Polynomial toolbox.

The computing time in all testing is measured by Matlab function stopwatch timer. All tests are performed under operating system Windows XP and in Matlab R14.

Before we verify our idea on computation of Diophantine equation we show that finding of solution using LAPACK is faster then standard MATLAB functions and the result precision is comparable. We demonstrate it on finding the solution of equation

$$\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B}, \tag{4.1}$$

where $\boldsymbol{A}$ is band matrix. This equation is computed through finding of solution of Diophantine equation by Sylvester method.

The random band matrices, using in the testing, are generated by created function
$>> \boldsymbol{A} = randbandmatrix(M, N, KL, KU),$
where *M, N, KL, KU* represent number of rows, number of columns, number of subdiagonals and number of superdiagonals, respectively.

- First test

  Our first test compares Matlab function left matrix divide and LAPACK function, *DGBSV*, for band square matrix in equation 4.1.LAPACK function is called through mex-file *genB*. The results are shown on the figure 4.1.

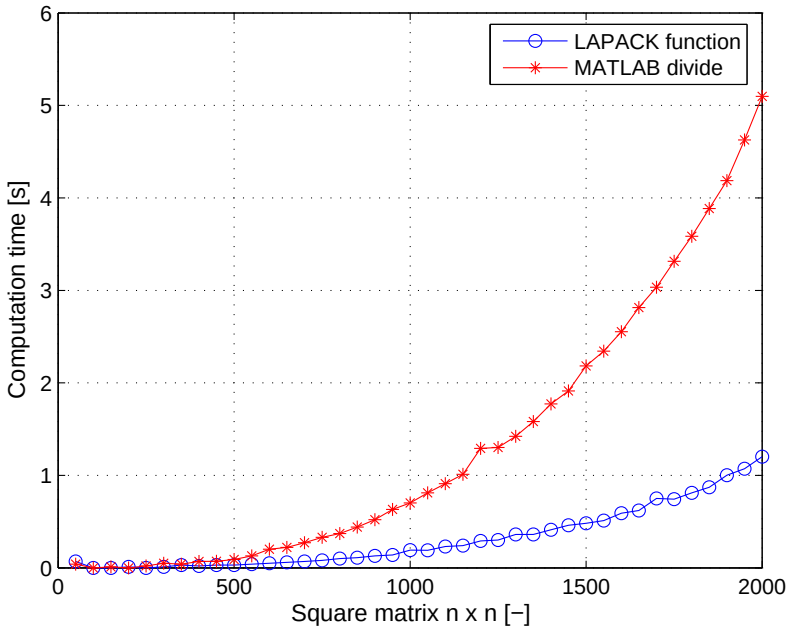

Figure 4.1: Comparison of Matlab function left matrix divide and LAPACK function, for band square matrix in equation 4.1

- Second test

  Our second test compares Matlab function left matrix divide and our function, *matrixMslN*, using LAPACK for band rectangular matrix in equation 4.1. The results are shown on the figure 4.2.

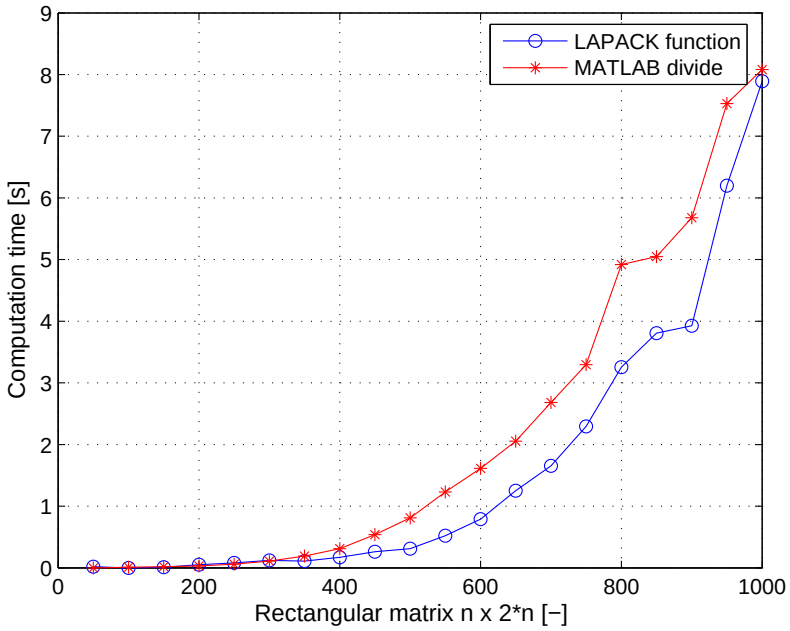

Figure 4.2: Comparison of Matlab function left matrix divide and our function for band rectangular matrix in equation 4.1

Now we compare Polynomial toolbox and our function for finding of the solution of polynomial Diophantine equation.

- Third test: scalar case of Diophantine equation

  Our third test compares Polynomial toolbox's function *axbyc* and the part for scalar case of Diophantine equation 1.1 of our funtion *axbycL* . The results are shown on the figure 4.3.

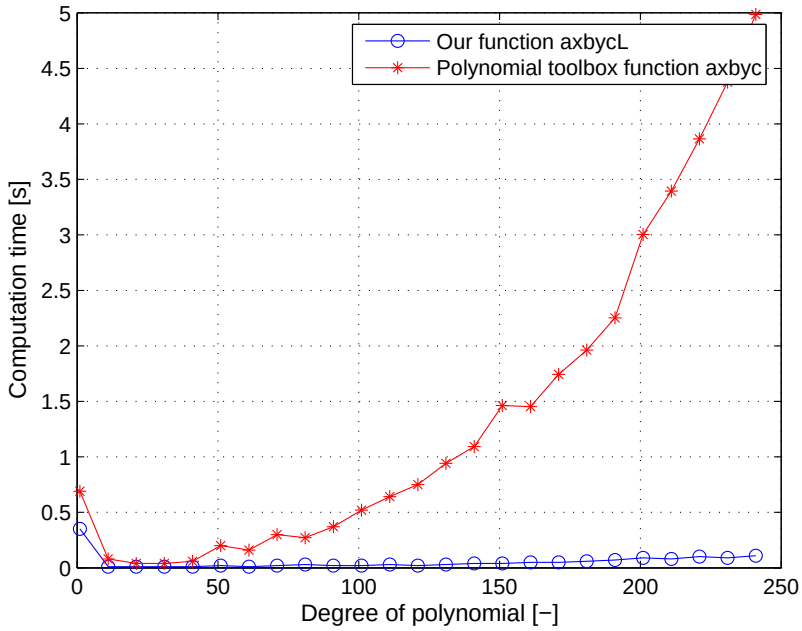

Figure 4.3: Comparison of Polynomial toolbox's function *axbyc* and our function *axbycL* for scalar case of Diophantine equation 1.1

- Fourth test: polynomial matrix Diophantine equation

  Our fourth test compares Polynomial toolbox's function *axbyc* and the second part of our funtion *axbycL* for matrix case of Diophantine equation 1.1. The results are shown on the figure 4.4.

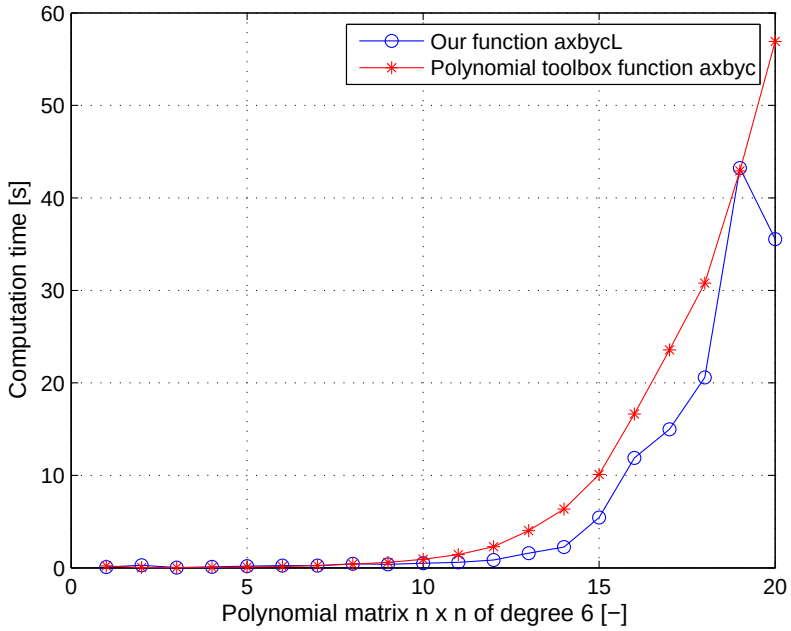

Figure 4.4: Comparison of Polynomial toolbox's function *axbyc* and our function *axbycL* for matrix case of Diophantine equation 1.1

Chapter 5

Conclusion

The main part of our work is devoted to the possibilities of improving the existing algorithms that solve Diophantine equation based on the knowledge of the matrix structure. We focus on solving Diophantine equation through Sylvester method, where special structure of Sylvester matrix appears. For acceleration of calculation we used external solver LAPACK. Because LAPACK does not contain functions for solving equations with matrices of special Sylvester structure, we used LAPACK functions for band matrices and we inserted Sylvester matrix in a general band matrix. We could do that because Sylvester matrix is special case of band matrix.

The achieved results of our work can be summarized as follows:

- Choice of the external solver

  We choose LAPACK as external solver. This choice based on a good co-operation between LAPACK and MATLAB and on knowledge that LAPACK contains functions for solving band matrices. When we closely study functions which are offered by LAPACK, we detect that LAPACK has function for solve the equation $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$, where $\boldsymbol{A}$ is square band matrix, and function for LU factorisation of general band matrix. When we solve the equation $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$, where $\boldsymbol{A}$ is general band matrix, we use function for LU factorisation and get matrix factors but after it we have to call other LAPACK's functions to find the solution of equation. These callings wrap our routines. but they are not optimized as well as LAPACK itself.

  This optimizing problem with our routine for general band matrix can be seen on the first and second test, see 4. Solving of the equation $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$, where $\boldsymbol{A}$ is square band matrix is computed directly through LAPACK function and it is faster then standard MATLAB function, as we assumed. But when $\boldsymbol{A}$ is general band matrix in the equation $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$, we computed this equation through our routines with

more then one LAPACK function the difference between computing times is not so big. Computing time of MATLAB function and computing time of our routine with LAPACK functions are comparable.

- Diophantine equation

  The usability of proposed improving of Sylvester method using external solver is realistic. It is possible to see on the third test and fourth test, see 4. Solving of Diophantine equation using function with external solver is faster then the function of Polynomial toolbox. For the scalar case the acceleration of method is obvious but for the matrix case there is not the acceleration so big.

- The future possibilities

  The idea of improving of method speed for solving problems in design control can be used also in other problems than solving of Diophantine equation. Some difficulty with this idea can be solved by choosing suitable external solver.

Bibliography

[1] HENRION, D. *Reliable Algorithms for Polynomial Matrices*.PhD. Thesis, Institute of Information Theory and Automation, Prague, 1998.

[2] GOLUB G. H. and LOAN, C. F. *Matrix Computations*.The Johns Hopkins Press Ltd. London, 1990.

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

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

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

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

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

[8] Ttd. Polyx. polyx [online]. 2005 [cit 2005-01-05] http://www.polyx.com

[9] DEMLOVÁ, M. and NAGY, J. *Algebra*. Praha:SNTL, 1985.

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

[11] KUČERA, V. *Discrete Linear Control (The Polynomial Equation Approach)*.Academia Prague, 1979.

[12] KUČERA, V. *Analysis and Design of Discrete Linear Control Systems*.Academia Prague, 1991.

[13] KUČERA, V. *Diophantine equations in control — a survey*.Automatica, 29(6):1361–1375, 1993.

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

[15] ANTSAKLIS P. J. and GAO Z. *Polynomial and Rational Matrix Interpolation: Theory and Control Applications*, Internetional Hournal on Control. Vol 58, pp 349 - 404, 1993.

[16] WOLFRAM S. *The Mathematica Book, Third Edition*. Cambridge University Press, 1996.

[17] HENRION D. and ŠEBEK M. *Numerical Methods for Polynomial Matrix Rank Evaluation*, Proceeding of the IFAC Conferenceon System Structure and Control, Nantes, France, July 1998.

[18] CHEN D. and MOLER C. *Symbolic Math Toolbox for Use with Matlab*. The MathWorks, Inc, 1993.

[19] ŠEBEK M. - HENRION D. - PEJCHOV8 S. - KWAKERNAAK H. *Numerical Methods for Zeros and Determinants of Polynomial Matrix*. in Proceeding of the 4thMediterrenean Symposium on New Direction in Control and Automation, (Chania, Crete, Greece), pp 473 - 477, IEEE-CSS, June 1996.

[20] HAVLENA, V. (1999). *Moderní teorie řízení - Doplňkové skriptum*. Vydavatelství ČVUT, Praha.

[21] HAVLENA, V. a ŠTECHA, J. (2000). *Moderní teorie řízení*. Vydavatelství ČVUT, Praha.

[22] ROUBAL, J. and PEKAŘ, J. Moderní teorie řízení [online]. Poslední revize 2003-07-01 [cit. 2003-07-01], http://dce.felk.cvut.cz/mtr/.