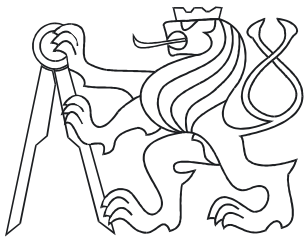
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ
KATEDRA ŘÍDICÍ TECHNIKY



DIPLOMOVÁ PRÁCE

MuPAD Library for Symbolic Computation with Polynomial Matrices

Petr Augusta

2005

Vedoucí: Ing. Zdeněk Hurák, Ph.D.
Centrum aplikované kybernetiky, Katedra řídicí techniky,
Fakulta elektrotechnická, České vysoké učení technické v Praze

Oponent: RNDr. Aleš Němeček
Katedra matematiky, Fakulta elektrotechnická,
České vysoké učení technické v Praze

Anotace

Práce popisuje nově vytvořenou a volně dostupnou knihovnu funkcí pro symbolické počítání s polynomy a polynomiálními maticemi nazvanou Polmat. Knihovna Polmat je vyvíjena pro MuPAD, snadno dostupný a výkonný počítačový algebraický systém a programovací jazyk. Samotný MuPAD neobsahuje funkce pro analýzu systémů a návrh regulačních obvodů a filtrů, a proto vznikla snaha tyto algoritmy vytvořit. Knihovna najde uplatnění ve výzkumu a vývoji tzv. algebraických metod návrhu regulátorů a filtrů. Implementovány byly funkce pro řešení lineárních rovnic s polynomy a polynomiálními maticemi, výpočet spektrální faktorizace polynomu a polynomiální matice a další. Dokument obsahuje úvod do počítání s polynomiálními maticemi a počítání s počítačovými algebraickými systémy, popis použití funkcí a procedur, vysvětlení užitých metod, demonstrativní příklady a výsledky zkušebních testů.

Abstract

This diploma thesis comes with a freely available library for symbolic computation with polynomials and polynomial matrices, named Polmat. The Polmat library is developed for MuPAD, which is an easily accessible efficient computer algebra system and programming language. MuPAD does not contain functions used for design and analysis of control systems, therefore, an endeavour to make them arose. Solvers for linear equations with polynomials and polynomial matrices, spectral factorization of polynomials and polynomial matrices and others have been implemented. This document contains introduction into polynomial matrix computation and basic usage of computer algebra systems, syntax of implemented functions and procedures, description of used methods, demonstrative examples and results of computational testing.

Contents

Chapter 1

Introduction

The polynomial methods, launched in 1970's from Czechoslovakia, are one of the two main used approaches to linear systems, the other being the so-called state-space methods. Within this framework, systems are described by input-output relations, i. e. the systems with single input and single output (SISO) by univariate polynomial fractions and the systems with multiple inputs and multiple outputs (MIMO) by left or right fractions of polynomial matrices. The procedures for analysis of properties of dynamical systems and controller design are based on manipulation with polynomials and polynomial matrices.

While the state-space methods, which operate with constant matrices, are included in many software libraries, there are a few packages that can handle polynomial matrices. The most complete library is no doubt the commercial Polynomial Toolbox for Matlab [44], developed by PolyX company. A few functions are implemented in Maple [43], Scilab [46]. Quite recently, some other packages for polynomial matrices computation for various software have been developed by the students of Czech Technical University in Prague: Mathematica package for polynomial matrices [21, 22] by Petr Kujan, package Polpack++ for C++ [6] by Leoš Halmo, Java package [33] by Michal Paděra, a library for TI-89/TI-92 programmable calculators [39] by Petr Štefko. These packages are based on numerical methods.

Numerical computation plays central role in industrial applications. Even the task of a controller design using polynomial methods is transformed into series of numerical operations with constant matrices, for which numerous software libraries are available (LAPACK, NAG, IMSL). The algorithms are based on e. g. Sylvester matrix methods, interpolation [8, 12] and there is endeavour to improve them all the time. However fast and efficient, these numerical algorithms are not always useful and reliable. The rounding errors introduced by finite precision arithmetics (mainly 64-bit floating point) take their price.

On the other hand, symbolic algorithms usually require more computing time and more storage then numeric ones. But symbolic computation gives the precise results and therefore even numerically unstable elementary operations can be performed without any risk of huge errors. In addition, computation with symbols is available. The role of symbolic algorithms can be found in helping with development of numerical algorithms and find out rounding errors and numerical unstability.

It is the aim of this work to give a package of functions for symbolic computation with polynomials and polynomial matrices. The created library is named Polmat and it is distributed

freely [1]. A computer algebra system MuPAD [45] was chosen for algorithms implementation. It provides a concept for object oriented programming, offers an interactive source code debugger and a powerfull visualization tool. A license of MuPAD versions for a student, a teacher, a lecturer, a representant of an educational institution or a private person is free. MuPAD is developed mainly at the University of Paderborn in Germany since 1990. Polmat has been implemented and tested in MuPAD 2.5.3.

In this document, the chapter **Polynomial Matrices Computation** gives the basic information about a computation with polynomials and polynomial matrices. The next chapter explains some fundamentals of **Computer Algebra Systems**. The chapter **MuPAD** is devoted to MuPAD software. A data type polmat and its definition is described in the chapter **New Data Type Polmat**. The implemented algorithms are described in the chapter **Algorithms**. In the chapter **Benchmarks** some tests and comparison with some others packages are made. The chapter **Applications in Control Design** gives some examples for using Polmat in the control theory.

Responses of Users

Stefan Wehmeier, MuPAD developer and researcher at Universität-Gesamthochschule Paderborn, together with Prof. Benno Fuchssteiner, head of MuPAD developers, have contacted us and offered cooperation and support.

Prof. Mirosław Majewski, Zayed University, organizer of MathPAD Conference, has invited us to MathPAD 2005, Toruń, Poland.

Ishan Pendharkar from Department of Electrical Engineering, Indian Institute of Technology Bombay, working in behavioral systems theory, uses Polmat for computing polynomial J-spectral factorization and matrix diagonalization.

Julian Stoev has contacted us and expressed his suggestions.

Chapter 2

Polynomial Matrix Computation

This chapter would like to give a short introduction into polynomial matrix computation. For detailed informations you can read e. g. [19, 24, 5, 25, 13, 23, 7, 37, 38].

2.1 Introduction

There are two main approaches to description of linear systems signals and control: state-space and polynomial description. The former uses four constant matrices and is often written as a system of differential (recurrence) and algebraic equations. The latter expresses linear model as a transfer function. *Single Input Single Output* (SISO) system is described by a fraction of two polynomials, i. e. ratio of Laplace transform of output and Laplace transform of input with zero initial conditions in continuous-time case and ratio of $\mathcal{Z}$-transform of output and $\mathcal{Z}$-transform of input with zero initial conditions in discrete-time case. *Multi Input Multi Output* (MIMO) system is described by a fraction of two polynomial matrices.

For instance, a system can be described by state-space methods as

$$\begin{aligned} \boldsymbol{x}(t+1) &= \begin{pmatrix} 0 & -\frac{1}{5} \\ 1 & \frac{6}{5} \end{pmatrix} \boldsymbol{x}(t) + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u(t) \\ y(t) &= \begin{pmatrix} \frac{4}{5} & \frac{13}{50} \end{pmatrix} \boldsymbol{x}(t) + \begin{pmatrix} 0 \end{pmatrix} u(t) \end{aligned}$$

where t is discrete time. The same system is described by polynomial methods as transfer function

$$Y(d) = \frac{\frac{4}{5}\,d - \frac{7}{10}\,d^2}{1 - \frac{6}{5}\,d + \frac{1}{5}\,d^2}\,U(d),$$

where d is complex variable. Obviously, the second method is more elegant than the first one. Polynomials and polynomial matrices arise naturally and have clear physical interpretation. This description is usually obtained from experimental identification and there is no need to introduce artificial variables called states. Sometimes the concept of state variable is very useful, because it enables us consider things that are going on inside the model, but sometimes this is undesirable if we want to focus on input-output behaviour only.

A following example shows controller design by polynomial methods.

Example 2.1

Consider the scheme in Fig. 2.1 and the unstable plant P, the controller C which are described by the following transfer functions

$$P(d) = \frac{b(d)}{a(d)} = \frac{\frac{4}{5}\,d - \frac{7}{10}\,d^2}{1 - \frac{6}{5}\,d + \frac{1}{5}\,d^2}, \qquad C(d) = \frac{y(d)}{x(d)}.$$

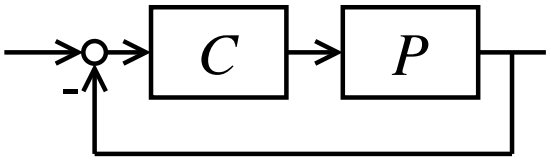


Figure 2.1: Basic block scheme of closed loop

Find the polynomials $x(d)$ and $y(d)$ such that closed-loop system is stable. A stability of closed loop depends on a polynomial $c(d)$ given by

$$c(d) = a(d)\,x(d) + b(d)\,y(d).$$

Choose any stable polynomial $c(d)$, for instance $c(d) = 1$, and solve the previous equation for $x(d)$ and $y(d)$

$$\left(1 - \frac{6}{5}\,d + \frac{1}{5}\,d^2\right)\,x(d) + \left(\frac{4}{5}\,d - \frac{7}{10}\,d^2\right)\,y(d) = 1.$$

This equation has infinite set of solutions, one of them is

$$\begin{aligned} x(d) &= 1 - \frac{238}{27}\,d \\ y(d) &= \frac{338}{27} - \frac{68}{27}\,d \end{aligned}$$

and the controller is

$$C(d) = \frac{y(d)}{x(d)} = \frac{68\,d - 338}{238\,d - 27}.$$

□

Simply, the controller design consists of the solution of a linear polynomial equation

$$a\,x + b\,y = c.$$

For MIMO systems, the process is analogous. Linear equations with polynomial matrices are solved. These equations may have one of the following forms

$$\begin{aligned} \boldsymbol{A}\,\boldsymbol{X} + \boldsymbol{B}\,\boldsymbol{Y} &= \boldsymbol{C} \\ \boldsymbol{A}\,\boldsymbol{X} + \boldsymbol{Y}\,\boldsymbol{B} &= \boldsymbol{C} \\ \boldsymbol{X}\,\boldsymbol{A} + \boldsymbol{Y}\,\boldsymbol{B} &= \boldsymbol{C}. \end{aligned}$$

In optimal control with quadratic control quality criterion a spectral factorization is computed. Then the equation

$$\boldsymbol{X}\boldsymbol{J}\boldsymbol{X}^* = \boldsymbol{B} \qquad \text{or} \qquad \boldsymbol{X}\boldsymbol{X}^* = \boldsymbol{B}$$

where $\boldsymbol{X}^*(s) = \boldsymbol{X}^{\mathrm{T}}(-s)$ in continuous-time case, $\boldsymbol{X}^*(z) = \boldsymbol{X}^{\mathrm{T}}(z^{-1})$ in discrete-time case is solved.

Very often in system analysis of synthesis, we need conversion of a polynomial matrix to one of several special forms, e.g., Smith form, Smith-McMillan form, Hermite form, row echelon form, etc.

Example 2.2

Consider the MIMO system [38]

$$\boldsymbol{G}(s) = \frac{1}{s\,(s+1)^2} \begin{pmatrix} 1 & (s+1)^2 & s\,(s+1)\,(s+2) \\ 0 & s\,(s+2) & 0 \\ 0 & 0 & 3\,s\,(s+1)\,(s+2) \end{pmatrix}.$$

Compute its zeros and poles. Smith-McMillan form of $\boldsymbol{G}(s)$ is

$$\frac{1}{s\,(s+1)^2} \begin{pmatrix} 1 & 0 & 0 \\ 0 & s\,(s+2) & 0 \\ 0 & 0 & 3\,s\,(s+1)\,(s+2) \end{pmatrix} = \begin{pmatrix} \dfrac{1}{s\,(s+1)^2} & 0 & 0 \\ 0 & \dfrac{s+2}{(s+1)^2} & 0 \\ 0 & 0 & \dfrac{3\,(s+2)}{(s+1)} \end{pmatrix}.$$

Poles, zeros are given by

$$\begin{aligned} s\,(s+1)^2 \cdot (s+1)^2 \cdot (s+1) &= 0, \\ (s+2) \cdot 3\,(s+2) &= 0, \end{aligned}$$

respectively. □

2.2 Methods

For all these mathematical routines the special algorithms are designed. They may be based on

- elementary operations on polynomial matrix,
- Sylvester matrix method,
- interpolation.

Let us introduce the corresponding definitions.

Definition 2.1 (*Elementary operations*) [19, 4]
Let $\boldsymbol{A}(\lambda)_{m,n}$ be a polynomial matrix. Elementary row (column) operations for polynomial matrix $\boldsymbol{A}(\lambda)$ have the form

(i) interchange of any two rows (columns),
(ii) addition to any row (column) of a polynomial multiple of any other row (column),
(iii) scaling any row (column) by any nonzero real or complex number.

□

The methods based on elementary operations on polynomial matrix are very efficient. Interesting in number of computations, these methods are the most effective of all. Because they are numerically unstable it is not possible use them in numerical algorithms.

Sylvester matrix methods are able to compute rather numerically than symbolically. Before we make clear what is Sylvester matrix we introduce a following definition.

Definition 2.2 (*Band matrix*)
Suppose a matrix

$$\boldsymbol{A} = (a_{ij}), \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n.$$

If there exists number p, $0 < p < \min(m, n)$, such that $a_{ij} = 0$ for all i, j for which $|i - j| > p$, an $\boldsymbol{A}$ is said to be *band matrix*. A number p is called *width of band*. □

Sylvester matrix [8]
Suppose a polynomial matrix equation

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

where $\boldsymbol{A}(s)_{m,n}$, $\boldsymbol{B}(s)_{m,p}$ are given polynomial matrices and $\boldsymbol{X}(s)_{n,p}$ is variable. The matrices can be written as

$$\begin{aligned}
\boldsymbol{A}(s) &= \boldsymbol{A}_0 + \boldsymbol{A}_1\, s + \boldsymbol{A}_2\, s^2 + \cdots + \boldsymbol{A}_{\partial \boldsymbol{A}}\, s^{\partial \boldsymbol{A}} \\
\boldsymbol{B}(s) &= \boldsymbol{B}_0 + \boldsymbol{B}_1\, s + \boldsymbol{B}_2\, s^2 + \cdots + \boldsymbol{B}_{\partial \boldsymbol{B}}\, s^{\partial \boldsymbol{B}} \\
\boldsymbol{X}(s) &= \boldsymbol{X}_0 + \boldsymbol{X}_1\, s + \boldsymbol{X}_2\, s^2 + \cdots + \boldsymbol{X}_{\partial \boldsymbol{X}}\, s^{\partial \boldsymbol{X}}.
\end{aligned}$$

Then we can write an equivalent equation

$$\underbrace{\begin{pmatrix}
\boldsymbol{A}_0 & & & \boldsymbol{0} \\
\boldsymbol{A}_1 & \boldsymbol{A}_0 & & \\
\vdots & \boldsymbol{A}_1 & \ddots & \\
\boldsymbol{A}_{\partial \boldsymbol{A}} & \vdots & & \boldsymbol{A}_0 \\
 & \boldsymbol{A}_{\partial \boldsymbol{A}} & & \boldsymbol{A}_1 \\
 & & \ddots & \vdots \\
\boldsymbol{0} & & & \boldsymbol{A}_{\partial \boldsymbol{A}}
\end{pmatrix}}_{\widehat{\boldsymbol{A}}}
\underbrace{\begin{pmatrix}
\boldsymbol{X}_0 \\ \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_{\partial \boldsymbol{X}}
\end{pmatrix}}_{\widehat{\boldsymbol{X}}}
=
\underbrace{\begin{pmatrix}
\boldsymbol{B}_0 \\ \boldsymbol{B}_1 \\ \vdots \\ \boldsymbol{B}_{\partial \boldsymbol{B}}
\end{pmatrix}}_{\widehat{\boldsymbol{B}}}$$

where $\widehat{\boldsymbol{A}}$ is an especial kind of band matrix called *Sylvester matrix*. □

A last mentioned method is the interpolation. It is not useful for symbolic computation. For its definition and algorithm see e. g. [8, 12, 21].

Chapter 3

Computer Algebra Systems

A *computer algebra system* is a type of software package that is used in manipulation of mathematical formulae. The primary aim of a computer algebra system is to solve tedious or difficult algebraic tasks. The computer algebra systems can be used interactively, the user enters some commands and the system evaluates them. The principal difference between a computer alebra system and a traditional calculator is the ability to deal with equations symbolically rather than numerically. In addition, most computer algebra systems can approximate solutions numerically and user can set the precision to the desired number of digits. The computer algebra systems often provide a programming language and tools for visualization and animation of mathematical data.

The beginning of development of computer algebra systems is jointed with comming of programming languages as Fortran and Lisp. The first program able to manipulate with symbolical expression SAIN (*Symbolic Automatic INtegration*) came in 1961, followed by FORMAC, MATHLAB, REDUCE, SCRATCHPAF, muMATH and others.

At present, there are many different computer algebra systems. Some are distributed commercially, others can be obtained for free. We distinguish *special purpose* and *general purpose* computer algebra systems. Special purpose systems can handle particular problems. For example, *GAP*, *LiE* deal with group theory, *CASA* and *GANITH* solve algebraic geometry problems, etc. The best-known general purpose systems are *MathCAD*, *Mathematica*, *Maple*, *Maxima*, *MuPAD*.

In what follows, we say more about Maple, Mathematica, Matlab and Scilab. Even though Matlab and Scilab do not belong strictly among CAS product, we consider them here.

3.1 Maple and Mathematica

Maple [43] and Mathematica [47] are ones of the most used computer algebra systems. They combine symbolic and numerical computing methods, provide graphical tool and programming language. Maple is distributed by Waterloo Maple, Inc. Mathematica is distributed by Wolfram Research, Inc. Mathematica package for polynomial matrix computation has been developed by Kujan [22].

3.2 Matlab and Polynomial Toolbox

Concerning technical universities, Matlab is the most widespread product. It enables fast numerical solvers, so needed in engineering. Matlab contains Symbolic Math Toolbox for symbolical computing. Matlab is trademark by The Mathworks, Inc.

The Polynomial Toolbox for Matlab [44] is a Matlab toolbox for polynomial matrix computation, developed by PolyX company. It offers e. g. new generation of numerical algorithms, continuous-time and discrete-time system and signal models based on polynomial matrix fractions, classical and robustness analysis for LTI systems and filters, classical and optimal design tools (pole placement, all stabilizing controllers, dead-beat, $\mathcal{H}_2$ and LQG), $\mathcal{H}_\infty$ optimization, conversion to and from LTI object of the Control System Toolbox. For futher informations see [44].

3.3 Scilab

Scilab [46] is mainly numerical package but there exists Scilab-MuPAD link (see Sec. 4.3), therefore, there is a short information about this project in this section.

Scilab is developed since 1990 by INRIA and ENPC. It is a scientific software package for numerical computations. It is distributed freely. Scilab aims at handling more complex objects than numerical matrices. The manipulation rational or polynomial transfer matrices is done by manipulating lists and typed lists which allows a natural symbolic representation of complicated mathematical objects such as transfer functions, linear systems or graphs.

Chapter 4

MuPAD

This chapter explains the basic use MuPAD. For more details see [32, 31, 45].

4.1 Introduction

MuPAD is an interactive general purpose computer algebra system in an integrated and open environment for symbolic and numeric computing. It has been developed at the University of Paderborn since 1990. MuPAD provides a concept for object oriented programming, offers an interactive source code debugger and a powerfull visualization tool called VCam. MuPAD kernel is implemented in C and C++, the MuPAD libraries are implemented in MuPAD's programming language. A user can write own MuPAD procedures and compile and link existing C/C++-code as *Dynamic Modules* at runtime. The MuPAD distributions for various operating systems exist.

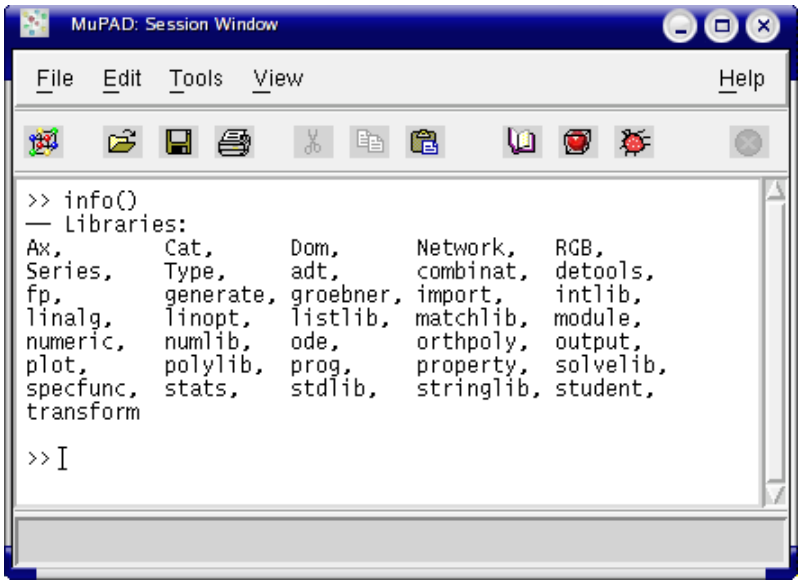


Figure 4.1: MuPAD

A license of some MuPAD versions for a student, a teacher, a lecturer, a representant of an educational institution or just a private person who is interested in doing mathematics is free, visit [45]. The free MuPAD versions, downloaded from [45], have a built-in memory limit of 6 megabytes. The registration removes it, see help for `register` command in MuPAD.

After starting program, you can enter an instruction and wait until MuPAD computes the result and prints it on the screen. For instance, to compute with numbers, you can type `1 + 3/2` and press <ENTER>. MuPAD displays

```
>> 1 + 3/2
                                    5/2
```

Most of MuPAD's mathematical knowledge is organized in libraries. A list of all available libraries can be printed by calling

```
>> info()
-- Libraries:
Ax,         Cat,       Dom,       Network,   RGB,
Series,     Type,      adt,       combinat,  detools,
fp,         generate,  groebner,  import,    intlib,
linalg,     linopt,    listlib,   matchlib,  module,
numeric,    numlib,    ode,       orthpoly,  output,
plot,       polylib,   prog,      property,  solvelib,
specfunc,   stats,     stdlib,    stringlib, student,
transform
```

You can get help by calling `?`. You can also type

```
>> ?linalg
>> ?linalg::stackMatrix
```

and get help for a library `linalg`, for a function `linalg::stackMatrix`, respectively, or choose Help in MuPAD menu.

The numbers, symbolic expressions, matrices, arrays, lists, equations, inequalities and more are called MuPAD objects. Every MuPAD object belongs to some data type, called the *domain type*. The types with capital letters as `DOM_INT`, `DOM_POLY` etc. correspond to domains provided by the MuPAD kernel. Domains as `Dom::Matrix()` were implemented in the MuPAD language. For full-range description of MuPAD objects read [32, Chap. 4].

By way of example we illustrate work with MuPAD.

Example 4.1

Student's exercise is to solve an equation

$$\frac{x+2}{7x+23} = \frac{x-2}{7(x+1)}.$$

But student see badly and copies it with mistaken second term in the numerator on left-hand side and with minus in the denominator on right-hand side. He solves this bad equation correctly and obtains right solution of given one. The question is what equation did he solve.

In MuPAD solve this tasks by following.

```
>> solve((x+2)/(7*x+23)=(x-2)/(7*(x+1)), x)

                                {-5}
>> solve((x+a)/(7*x+23)=(x-2)/(7*(x-1)), x)

              / { 7 a - 46 }                                  \
     piecewise| { -------- } if a <> 16/7, {} if a = 16/7 |
              \ { 7 a - 16 }                                  /
>> solve((7*a-46)/(7*a-16)=-5, a)

                                 {3}
```

A student solved the equation

$$\frac{x+3}{7x+23} = \frac{x-2}{7(x-1)}.$$

□

Example 4.2

Solve a linear optimization problem

$$\max\left\{ 2x-3y \;\middle|\; \begin{array}{rcl} x+2y & \geq & 6 \\ y-x & \leq & 3 \\ x+y & \leq & 10 \end{array} \;,\; x, y \geq 0 \right\}.$$

Get a numeric solution via `linopt::maximize` at first.

```
>> linopt::maximize([{x+2*y>=6, y-x<=3, x+y<=10}, 2*x-3*y,
      NonNegative])

                   [OPTIMAL, {y = 0, x = 10}, 20]
```

You can also use of MuPAD's tool for vizualization of mathematical data to solve our task by graphic method.

```
>> k:=[[x+2*y>=6, y-x<=3, x+y<=10], 2*x-3*y, NonNegative]:
>> g:=linopt::plot_data(k, [x, y]):
>> plot2d(g)
```

After performing a last command, MuPAD opens a window in Fig. 4.2. □

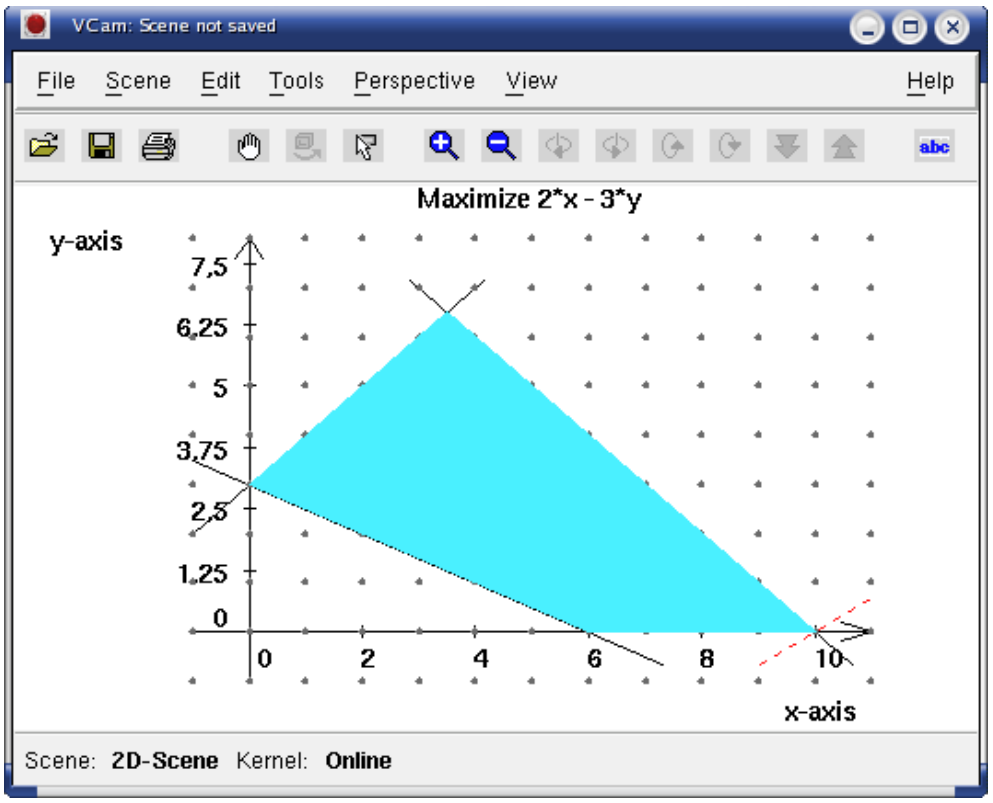


Figure 4.2: Graphic method for solving a linear optimalization problem

4.2 MuPAD Packages

MuPAD provides a programming language in which the user can implement complex algorithms. The procedures can be defined via `proc-end_proc` and can be joined to packages [32]. All functions, contained in a package, are avaible after calling command

```
package("$PATH")
```

where `$PATH` denotes package directory path. In Linux, place packages to directory

```
~/.mupad/packages
```

for easy loading.

Example 4.3

You can load a Polmat library by, for example,

```
>> package("polmat")

              "Library Polmat successfully loaded."
```

□

4.3 MuPAD-Scilab Link

The MuPAD-Scilab link [29] is the link between MuPAD and the numerical Scilab package [46]. Using this link, data can be exchanged between MuPAD and Scilab and some Scilab commands can be executed from inside a MuPAD session.

You can perform numerical computing by Scilab in MuPAD, as in following example.

Example 4.4

```
>> scilab::start()

                                  TRUE
>> scilab::sqrt(2)

                              1.414213562
```

□

Chapter 5

New Data Type Polmat

The Polmat library uses MuPAD domains and data types for definition its objects and combines them. Polmat works with four rings, with polynomials, polynomial matrices, rational functions and rational matrices. You can define a new object by function `polmat::new`, or shortly `polmat`. The function chooses object data type according to input data.

The domains `DOM_POLY` is used for definition of a polynomial. It is a special data type which is implemented with some kernel functions and makes computation with polynomials more efficient. The analogous data type for polynomial matrices is missing. Therefore, a polynomial matrix is an element of domain type

```
Dom::Matrix(Dom::DistributedPolynomial([x],
Dom::ExpressionField(normal, iszero@normal), LexOrder))
```

where `x` denotes an indeterminate. For better manipulation with polynomial matrices, command `polmat::poly:=Dom::DistributedPolynomial` is performed.

For computing with the polynomials in control theory, we often need use the polynomials with negative exponents, for example $a(z) = 2z + 1 + 2z^{-1}$. Such polynomials are defined as `DOM_EXPR` and a matrix of such polynomials is object of domain `Dom::Matrix()`.

Our approach has only one trouble. There is not compability of scalar polynomial with 1 by 1 polynomial matrix or a term of a polynomial matrix, because any matrix domain type is not compatible with type `DOM_POLY` in MuPAD.

The description of a command `polmat` follows.

5.1 Definition of a New Object

As it was said, `polmat::new` defines a new object – a polynomial, a polynomial matrix, a rational function or a rational matrix.

Call(s)

```
a:=polmat(p<, x>)
A:=polmat(m, x)
```

Parameters and Options

`p` – a polynomial or an expression
`m` – an element of a domain `DOM_LIST`, see the Sec. 5.2
`x` – an indeterminate

Return Values

`a` – an element of a domain `DOM_EXPR`, `DOM_POLY`, `DOM_INT`, ...
`A` – an element of a domain `Dom::Matrix(polmat::poly)` or `Dom::Matrix()`

5.2 Examples

The creation of new objects is best seen via a simple examples.

Example 5.1

Create a polynomial and the polynomial matrices

$$a(x) = 10\,c\,x^2 + x, \qquad \boldsymbol{A}(d) = \begin{pmatrix} d & 1-d \\ 5\,d^2 & d^2 \end{pmatrix}, \qquad \boldsymbol{B}(x) = \begin{pmatrix} x + x^{-1} & 1 + x^{-2} \\ 5\,x + 1 & 2 \end{pmatrix}.$$

```
>> a:=polmat(x+10*c*x^2, x)

                                    2
                        poly((10 c) x  + x, [x])
>> A:=polmat([[d, -d+1],[5*d^2, d^2]], d)
                           +-                -+
                           |    d,  - d + 1   |
                           |                  |
                           |     2      2     |
                           |  5 d ,    d      |
                           +-                -+
>> B:=polmat([[x+x^-1, x^-2],[5*x+1, 2]], x)
                           +-              -+
                           |       1    1   |
                           |   x + -,   --  |
                           |       x    2   |
                           |            x   |
                           |                |
                           |  5 x + 1,  2   |
                           +-              -+
```

□

Example 5.2

It is available to define a number as constant or as polynomial.

```
>> a:=polmat(3)

                                    3
>> domtype(%)

                                 DOM_INT
>> a:=polmat(3, x)

                               poly(3, [x])
>> domtype(%)

                                 DOM_POLY
```

□

Example 5.3

In control theory, the discrete-time systems are described by polynomials in z or in $z^{-1} = d$. In the second case, you can resolve if an indeterminate will be z or d.

```
>> c:=polmat(1/z + 3/z^2)

                                1    3
                                - + --
                                z    2
                                     z
>> c:=polmat(1/z + 3/z^2, 1/z)

                                   2
                           poly(3 d  + d, [d])
```

□

Chapter 6

Algorithms

This chapter serves the algorithms description. We mention the algorithms for a greatest common divisor, an adjoint of a matrix, the equations solvers, a spectral factorization, the matrix decompositions and reductions, a rank of a matrix, the Routh table, the Schmidt pairs, the Nehari problem, a division of the polynomials and a minimal realization. You can read more informations, some examples and the description of all algorithms implemented in Polmat in [2].

6.1 Greatest Common Divisor

For the polynomial computation, greatest common divisor (GCD) of polynomials is one of the most important algorithms. In Polmat, it is computed by function `polmat::gcd`.

Call(s)

```
[d, p, q, r, s] := polmat::gcd(a, b, x)
[D, P, Q, R, S] := polmat::gcd(A, B, x<, "left">)
[D, P, Q, R, S] := polmat::gcd(A, B, x<, "right">)
```

Parameters and Options

`a, b` – the polynomials of domain `DOM_POLY`
`A, B` – the matrices of a type `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

Return Values

`d, p, q, r, s` – the polynomials of domain `DOM_POLY`
`D` – a matrix of a type `Dom::Matrix(polmat::poly)`
`P, Q, R, S` – the matrices of a type `Dom::Matrix()`

Details

Scalar version of the algorithm solves a system of equations

$$\begin{aligned} ap - bq &= d \\ ar - bs &= 0, \end{aligned}$$

where a, b are given polynomials, p, q, r, s, d are variables and d is GCD. Computation method is based on Euclidean algorithm.

Algorithm for polynomial matrices finds left, right GCD of the polynomial matrices, solves a system of equations

$$\begin{aligned} \boldsymbol{AP}+\boldsymbol{BQ} &= \boldsymbol{D} \\ \boldsymbol{AR}+\boldsymbol{BS} &= \boldsymbol{0}, \end{aligned} \qquad \begin{aligned} \boldsymbol{PA}+\boldsymbol{QB} &= \boldsymbol{D} \\ \boldsymbol{RA}+\boldsymbol{SB} &= \boldsymbol{0}, \end{aligned}$$

respectively, where $\boldsymbol{A}$, $\boldsymbol{B}$ are given polynomial matrices, $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{S}$ are variables and $\boldsymbol{D}$ is GCD.

These algorithms are based on elementary operations on polynomial matrix. For full description the both GCD algorithms see [23].

6.2 Adjoint, Inverse, Determinant and Rank

Algorithms for computation of adjoint, inverse, determinant and rank of a polynomial matrix are contained in this section.

6.2.1 Adjoint of a Matrix and Determinant

Adjoint of a matrix adj$\boldsymbol{A}$ and the determinant det$\boldsymbol{A}$ of a matrix $\boldsymbol{A}$ are computed by function `polmat::adj`.

Call(s)

```
[adj, det] := polmat::adj(A, x)
```

Parameters and Options

`A` – an element of a domain `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

Return Values

`adj` – an element of a domain `Dom::Matrix(polmat::poly)`
`det` – an element of a domain `polmat::poly`

Details

Algorithm is based on elementary operations on polynomial matrix. For its full description read [23].

Also function `polmat::det` for computing matrix determinant is available, see [2].

6.2.2 Inverse of a Matrix

Inverse of a matrix is returned by function `polmat::inverse`.

Call(s)

```
B := polmat::inverse(A, x)
```

Parameters and Options

`A` – an element of a domain `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

Return Values

`B` – an element of a domain `Dom::Matrix(polmat::poly)`

Details

Inverse $\boldsymbol{A}^{-1}$ for a matrix $\boldsymbol{A}$ is computed by using adjoint and determinant of a matrix by

$$\boldsymbol{A}^{-1} = \frac{\mathrm{adj}\boldsymbol{A}}{\det\boldsymbol{A}}.$$

6.2.3 Rank of a Polynomial Matrix

Rank is computed by function `polmat::rank`.

Call(s)

```
r := polmat::rank(A, x<, horn>)
r := polmat::rank(A, x<, sylv>)
```

Parameters and Options

`A` – a matrix of a domain `Dom::Matrix(polmat::poly)`
`x` – an indeterminate
horn – Horner scheme method is used
sylv – Sylvester matrix algorithm used

Return Values

`r` – a nonnegative integer number

Details

There are two methods for computing rank of a polynomial matrix [8, 4]. The first one uses matrix evaluation by *Horner scheme* [49, 4], the second one is *Sylvester matrix algorithm* [8]. Mentioned methods are performed by the macros `polmat::value` and `polmat::gensylv` [2].

Optional parameter (*horn/sylv*) determines what method will be used. The default is Horner scheme method.

6.3 Solution of Linear Equations with Polynomials, Polynomial Matrices

Solvers equations with polynomials and polynomial matrices are described in this section.

6.3.1 AXBYC Equation

Solution of linear equation with polynomials, polynomial matrices

$$a\,x + b\,y = c, \tag{6.1}$$
$$\boldsymbol{A}\,\boldsymbol{X} + \boldsymbol{B}\,\boldsymbol{Y} = \boldsymbol{C}, \tag{6.2}$$

respectively, where a, b, c are given polynomials, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ are given polynomial matrices and x, y, $\boldsymbol{X}$, $\boldsymbol{Y}$ are variables is returned by `polmat::axbyc`.

Call(s)

```
[X,Y] := polmat::axbyc(a, b, c, x<, "x-minimal">)
[X,Y] := polmat::axbyc(a, b, c, x<, "y-minimal">)
[X,Y] := polmat::axbyc(a, b, c, x<, "degx<dega">)
[X,Y] := polmat::axbyc(A, B, C, x)
```

Parameters and Options

`a, b, c` – the polynomials
`A, B, C` – the polynomial matrices of type `polmat::matrix`
`x` – an indeterminate

Return Values

`X, Y` – an element of a domain `DOM_POLY` or `Dom::Matrix(polmat::poly)`

Details

General solution of (6.1) is

$$\begin{aligned} x &= p\frac{c}{d} + r\,t \\ y &= q\frac{c}{d} + s\,t, \end{aligned}$$

where polynomials d, p, q, r, s are solution of GCD algorithm and t is an arbitrary one.

General solution (6.2) is

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{P}_2\boldsymbol{C}_2 + \boldsymbol{R}_2\boldsymbol{T} \\ \boldsymbol{Y} &= \boldsymbol{Q}_2\boldsymbol{C}_2 + \boldsymbol{S}_2\boldsymbol{T}, \end{aligned}$$

where $\boldsymbol{T}$ is an arbitrary matrix and

$$\begin{aligned} \boldsymbol{C} &= \boldsymbol{D}_2\boldsymbol{C}_2, \\ \boldsymbol{A}\boldsymbol{P}_2 + \boldsymbol{B}\boldsymbol{Q}_2 &= \boldsymbol{D}_2 \\ \boldsymbol{A}\boldsymbol{R}_2 + \boldsymbol{B}\boldsymbol{S}_2 &= \boldsymbol{0}. \end{aligned} \tag{6.3}$$

If the parameter `"x-minimal"` is given, the equation is solved for x-minimal solution x_1, y_1. If the parameter `"y-minimal"` is given, the equation is solved for y-minimal solution x_2, y_2. These algorithms are described in [14].

If the parameter `"degy<dega"` is given, the equation is solved for $\partial y_3 < \partial a$. General solution is

$$\begin{aligned} x_3 &= x_2 + \frac{b}{(a,\ b)}\,t \\ y_3 &= y_2 - \frac{a}{(a,\ b)}\,t, \end{aligned}$$

where an arbitrary polynomial t must satisfy $\partial t < \partial(a,\ b)$.

6.3.2 AXYBC Equation

Solution of linear equations of polynomials, polynomial matrices

$$\begin{aligned} a\,x + y\,b &= c \\ \boldsymbol{A}\boldsymbol{X} + \boldsymbol{Y}\boldsymbol{B} &= \boldsymbol{C}, \end{aligned}$$

respectively, where a, b, c are given polynomials, $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are given polynomial matrices and x, y, $\boldsymbol{X}, \boldsymbol{Y}$ are variables is returned by `polmat::axybc`.

Call(s)

```
[X, Y] := polmat::axybc(a, b, c, x)
[X, Y] := polmat::axybc(A, B, C, x)
```

Parameters and Options

`a, b, c` – the polynomials
`A, B, C` – the polynomial matrices of type `polmat::matrix`
`x` – an indeterminate

Return Values

`X, Y` – the elements of a domain `DOM_POLY` or `Dom::Matrix(polmat::poly)`

6.3.3 XAYBC Equation

Solution of linear equations of polynomials or polynomial matrices

$$\begin{aligned} x\,a + y\,b &= c, \\ \boldsymbol{X}\,\boldsymbol{A} + \boldsymbol{Y}\,\boldsymbol{B} &= \boldsymbol{C}, \end{aligned} \tag{6.4}$$

respectively, where a, b, c are given polynomials, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ are given polynomial matrices and x, y, $\boldsymbol{X}$, $\boldsymbol{Y}$ are variables is returned by `polmat::xabyc`.

Call(s)

```
[X, Y] := polmat::xabyc(a, b, c, x)
[X, Y] := polmat::xabyc(A, B, C, x)
```

Parameters and Options

`a, b, c` – the polynomials
`A, B, C` – the polynomial matrices of type `polmat::matrix`
`x` – an indeterminate

Return Values

`X, Y` – the elements of a domain `DOM_POLY` or `Dom::Matrix(polmat::poly)`

Details

General solution of (6.4) is

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{C}_1\,\boldsymbol{P}_1 + \boldsymbol{T}\,\boldsymbol{R}_1 \\ \boldsymbol{Y} &= \boldsymbol{C}_1\,\boldsymbol{Q}_1 + \boldsymbol{T}\,\boldsymbol{S}_1, \end{aligned} \tag{6.5}$$

where $\boldsymbol{T}$ is an arbitrary matrix and

$$\begin{aligned} \boldsymbol{C} &= \boldsymbol{C}_1\,\boldsymbol{D}_1, \\ \boldsymbol{P}_1\,\boldsymbol{A} + \boldsymbol{Q}_1\,\boldsymbol{B} &= \boldsymbol{D}_1 \\ \boldsymbol{R}_1\,\boldsymbol{A} + \boldsymbol{S}_1\,\boldsymbol{B} &= \boldsymbol{0}. \end{aligned}$$

6.3.4 AXBYCD Equation

Solution of linear equations of polynomials

$$a\,\bar{x} + \bar{b}\,y = c + \bar{d},$$

where a, b, c, d are given polynomials and x, y are variables is returned by `polmat::axbycd`.

Call(s)

```
[X, Y] := polmat::axbycd(a, b, c, d_c, x<, "x-minimal">)
[X, Y] := polmat::axbycd(a, b, c, d_c, x<, "y-minimal">)
```

Parameters and Options

`a, b, c` – the polynomials, the expressions or the numbers
`d_c` – an expression
`x` – an indeterminate

Return Values

`X, Y` – the elements of a domain `DOM_POLY`

Details

If the parameter `"x-minimal"` is given, the equation is solved for x-minimal solution. This parameter is default. If the parameter `"y-minimal"` is given, the equation is solved for y--minimal solution. The algorithm is described in [16].

6.3.5 AXXAB Equation

Solution of symmetric linear equations of polynomials

$$\bar{a}\,x + a\,\bar{x} = b,$$

where a is given polynomial, b is given two-sided one and x is variable is returned by function `polmat::axxab`.

Call(s)

```
X := polmat::axxab(a, b, x)
```

Parameters and Options

`a, b` – the polynomials
`x` – an indeterminate

Return Values

`X` – an element of a domain `DOM_POLY`

6.4 Spectral Factorization

In various optimal control techniques, spectral factorization is a crucial computational step. Algorithm for polynomial spectral factorization and algorithm for polynomial matrix J-spectral factorization is performed by function `polmat::spf`.

Call(s)

```
[X, n, eps] := polmat::spf(p, x)
[P, J] := polmat::spf(Z, x)
```

Parameters and Options

`p` – the polynomials of domain `DOM_POLY`
`Z` – a polynomial matrix of type `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

Return Values

`X` – a polynomial of domain `DOM_POLY`
`n` – an integer number
`eps` – a real number
`P, J` – the polynomial matrices of type `Dom::Matrix(polmat::poly)`

Details

Polynomial spectral factorization problem means to find a polynomial x satisfying

$$x\,\bar{x} = b,$$

where b is given polynomial and $\bar{x}(s) = x(-s)$ in continue-time case, $\bar{x}(d) = x(d^{-1})$ in discrete-time case. The implemented algorithm for polynomial spectral factorization is based on a Newton-Raphson iterative scheme and on the macro `polmat::axxab`. For details see [40, 41].

The Newton-Raphson method is stopped when $\mathrm{norm}(x_i - x_{i-1})$ is less then a value of `polmat::spf::tol` or a number of iteration is a value of `polmat::spf::noi`. The default value of `polmat::spf::tol` is 10^{-8}. The default value of `polmat::spf::noi` is 30.

The scalar version of function `polmat::spf` returns a polynomial x, number of performed iterations n and precision ε.

Polynomial matrix version of spectral factorization algorithm is based on a diagonalization [13, 26]. For input polynomial matrix $\boldsymbol{Z}$ the matrices $\boldsymbol{P}$ and $\boldsymbol{J}$ are computed that

$$\boldsymbol{Z} = \bar{\boldsymbol{P}}\,\boldsymbol{J}\,\boldsymbol{P},$$

where $\bar{\boldsymbol{P}}(s) = \boldsymbol{P}^{\mathrm{T}}(-s)$ and the constant matrix $\boldsymbol{J}$ has a form

$$\begin{pmatrix} \boldsymbol{I}_1 & 0 & 0 \\ 0 & -\boldsymbol{I}_2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

The algorithm for discrete-time case is not available.

6.5 Matrix Reductions and Decompositions

This chapter refers to some matrix decompositions and the reductions to some forms. Algorithms for computing Hermite form, Smith form, Popov form, a reduction by a right and a left divisor and others are presented.

6.5.1 Smith Form

An input matrix is reduced by elementary column and row operations to Smith form [23, 19] by function `polmat::smithForm`. The algorithm computes matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ so that

$$\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{Q} = \mathrm{diag}_{l,\,m}\,[a_1,\ a_2,\ \ldots,\ a_r,\ 0,\ \ldots,\ 0]\,,$$

where $\boldsymbol{A}_{l,\,m}$ is a given matrix. For more details see [23].

Call(s)

```
[P, B, Q, r] := polmat::smithForm(A, x)
```

Parameters and Options

`A` – a matrix of a type `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

Return Values

`P, B, Q` – the matrices of a domain `Dom::Matrix` (`B`= $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{Q}$),
`r` – an integer number

6.5.2 Hermite Form

Matrix reduction to Hermite form [19] is performed by function `polmat::hermiteForm`. It computes matrices $\boldsymbol{P}$, $\boldsymbol{Q}$ so that $\boldsymbol{PA}$, $\boldsymbol{AQ}$ is row, column Hermite form, respectively, where $\boldsymbol{A}$ is given matrix. The computing method is based on elementary operations on polynomial matrix. If the parameter *col* is given column Hermite form is computed else row Hermite form is computed.

Call(s)

```
[X, T] := polmat::hermiteForm(A, x<, col>)
[X, T] := polmat::hermiteForm(A, x<, row>)
```

Parameters and Options

`A` – a matrix of a type `Dom::Matrix(polmat::poly)`

`x` – an indeterminate

Return Values

`X, T` – the matrices of a domain `Dom::Matrix`, Hermite form and a transformation matrix

6.5.3 Popov Form

Matrix reduction to Popov form [19, 30] is computed by function `polmat::popovForm`. For definition Popov form and algorithm see [19]. If the parameter *col* is given column Popov form is computed else row Popov form is computed. An input matrix must be column, row reduced, respectively.

Call(s)

```
X := polmat::popovForm(A, x<, col>)
X := polmat::popovForm(A, x<, row>)
```

Parameters and Options

`A` – a matrix of a type `Dom::Matrix(polmat::poly)`

`x` – an indeterminate

Return Values

`X` – a matrix of a domain `Dom::Matrix`

6.5.4 Right and Left Decomposition

Right or left matrix decomposition

$$\boldsymbol{S} = \boldsymbol{B}_1 \boldsymbol{A}_2^{-1}, \qquad \boldsymbol{S} = \boldsymbol{A}_1^{-1} \boldsymbol{B}_2,$$

respectively, is returned by function `polmat::decomp`. An input matrix $\boldsymbol{S}$ must be given as

$$\boldsymbol{S} = \frac{\boldsymbol{B}}{a}$$

where $\boldsymbol{B}$ is l by m matrix and a is polynomial.

For right, left decomposition, the algorithm computes left GCD of $\boldsymbol{A}_l$ and $\boldsymbol{B}$, right GCD of $\boldsymbol{A}_m$ and $\boldsymbol{B}$, respectively, where $\boldsymbol{A}_l = a\,\boldsymbol{I}_l, \boldsymbol{A}_m = a\,\boldsymbol{I}_m$. For full-range description see [23].

Call(s)

```
[A2, B1] := polmat::decomp(B, a, x<, "right">)
[A1, B2] := polmat::decomp(B, a, x<, "left">)
```

Parameters and Options

`B` – a matrix of a type `Dom::Matrix(polmat::poly)`
`p` – a polynomial of a domain `DOM_POLY`
`x` – an indeterminate

Return Values

`A1, B1, A2, B2` – the matrices of a domain `Dom::Matrix`

6.5.5 Matrix Reduction Test

Function `polmat::isreduced` tests whether a matrix is column or row reduced.

Call(s)

```
polmat::isreduced(m, x<, "col">)
polmat::isreduced(m, x<, "row">)
```

Parameters and Options

`m` – a matrix of a type `Dom::Matrix(polmat::poly)`
`x` – an indeterminate
`"col"` – a column reduction test, default
`"row"` – a row reduction test

Return Values

a value of a domain `DOM_BOOL`

6.5.6 Reduction by a Right and a Left Divisor

Reduction by right or left divisor is made by function `polmat::reduce`. The algorithm computes matrix $\boldsymbol{C}_1, \boldsymbol{C}_2$ so that

$$\boldsymbol{C} = \boldsymbol{C}_1 \boldsymbol{D}_1, \qquad \boldsymbol{C} = \boldsymbol{D}_2 \boldsymbol{C}_2,$$

where $\boldsymbol{D}_1, \boldsymbol{D}_2$ is right, left divisor of given matrix $\boldsymbol{C}$, respectively. Presumption is that divisors are computed by function `polmat::gcd` and they are in triangular form. This algorithm is described in [23].

Call(s)

```
C1 := polmat::reduce(C, D1, x<, "right">)
C2 := polmat::reduce(C, D2, x<, "left">)
```

Parameters and Options

`C, D1, D2` – a matrix of a type `Dom::Matrix(polmat::poly)`
`x` – an indeterminate
`"right"` – a reduction by right divisor, default
`"left"` – a reduction by left divisor

Return Values

`C1, C2` – a matrix of a type `Dom::Matrix(polmat::poly)`

6.5.7 Column and Row Reduction

Function `polmat::reduce` returns column, row reduced matrix for an input matrix $\boldsymbol{A}$. Algorithm computes matrix $\boldsymbol{A}_r, \boldsymbol{A}_c$, and unimodular matrix $\boldsymbol{U}, \boldsymbol{V}$ that

$$\boldsymbol{A}_r = \boldsymbol{U}\boldsymbol{A}, \qquad \boldsymbol{A}_c = \boldsymbol{A}\boldsymbol{V},$$

respectively. The method is based on elementary operations on polynomial matrix. The default method is row reduction.

Call(s)

```
[Ar, U] := polmat::reduce(A, x,<, row>)
[Ac, V] := polmat::reduce(A, x,<, col>)
```

Parameters and Options

`A` – a matrix of a domain `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

Return Values

`Ar, Ac, U, V` – the matrices of a domain `Dom::Matrix(polmat::poly)`

6.6 Routh Table and Its Offer in Addition to Stability

This chapter deals with the algorithms derived from [35]. There is shown that *Routh table* [35, 38] can be used to construct an orthonormal basis in the space of strictly proper rational function with a common stable denominator and so can be used to compute the $\mathcal{H}_2$ norm, the Hankel approximation and the Nehari problems.

In following, consider a strictly proper stable signal or system

$$G(s) = \frac{b(s)}{a(s)}. \tag{6.6}$$

6.6.1 Routh and Augmented Routh Table

Routh table is constructed by function `polmat::routh`. *Augmented Routh table* [35] is constructed by function `polmat::augRouth`.

Call(s)

```
R := polmat::Routh(a, x)
[R, Q1, Q2] := polmat::augRouth(b, a, x)
```

Parameters and Options

`a, b` – the polynomials of a type `DOM_POLY`
`x` – an indeterminate

Return Values

`R, Q1, Q2` – the matrices of a domain `Dom::Matrix()`

6.6.2 $\mathcal{H}_2$ Norm

$\mathcal{H}_2$ norm of (6.6) is computed by function `polmat::h2norm`.

Call(s)

```
h := polmat::h2norm(b, a, x)
```

Parameters and Options

`a, b` – the polynomials of a type `DOM_POLY`
`x` – an indeterminate

Return Values

`h` – a number or an expression

6.6.3 Schmidt Pairs

Function `polmat::schmidtPair` computes Schmidt pairs for (6.6).

Call(s)

```
schp := polmat::schmidtPair(b, a, x)
```

Parameters and Options

`a, b` – the polynomials of a type `DOM_POLY`
`x` – an indeterminate

Return Values

`schp` – a table of arrays of an expression

6.6.4 Nehari Problem

Function `polmat::nehari` finds $Q(s) \in \mathcal{H}_\infty$ to minimize $\|G(-s) - Q(s)\|_\infty$.

Call(s)

```
F := polmat::nehari(b, a, x)
```

Parameters and Options

`a, b` – the polynomials of a type `DOM_POLY`
`x` – an indeterminate

Return Values

`F` – an expression

6.7 Division of Polynomials

Functions for computing division of polynomials are contained in this section. The first function returns a quotient and a reminder, the second one computes a long division. Both functions are described in [23].

6.7.1 Division of Two Polynomials

Function `polmat::divide` divides polynomials a and b and returns a quotient q and a reminder r. We can write

$$q = \frac{a}{b} - \frac{r}{b}.$$

Call(s)

```
[q, r] := polmat::divide(a, b, x)
```

Parameters and Options

`a, b` – the polynomials of a domain `DOM_POLY`
`x` – an indeterminate

Return Values

`q, r` – the polynomials of a domain `DOM_POLY`

6.7.2 Long Division

Function `polmat::longdivide` makes expansion of a transfer function to series. The transfer function may be given as two polynomials a and b or as expression c. The first k members of series are computed.

Call(s)

```
q := polmat::longdivide(a, b, x, k)
q := polmat::longdivide(c, x, k)
```

Parameters and Options

`a, b` – the polynomials of a domain `DOM_POLY`
`c` – an expression
`x` – an indeterminate
`k` – a nonnegative integer number

Return Values

`q` – a polynomial of a domain `DOM_POLY`

6.8 Minimal Realization

The last algorithm described in this paper deals with a state-space realization. Consider system defined by

$$\boldsymbol{S} = \frac{\boldsymbol{B}}{a}. \tag{6.7}$$

Function `polmat::minreal` computes four constant matrices, which are a minimal realization of $\boldsymbol{S}$. This algorithm is based on [23].

Call(s)

```
[F, G, H, J] := polmat::minreal(B, a, x)
```

Parameters and Options

`B` – a matrix of a domain `Dom::Matrix(polmat::poly)`
`a` – the polynomials of a domain `DOM_POLY`
`x` – an indeterminate

Return Values

`F, G, H, J` – the matrices of a domain `Dom::Matrix`

Chapter 7

Benchmarks

We compare the Polmat library with a MuPAD library `linalg` and some polynomial matrices computation oriented packages. We are interested in the computing time. Polmat performs a symbolic computation, while, for instance, the Polynomial Toolbox for Matlab is a numerical package. We presume the numeric packages as Polynomial Toolbox or Scilab are faster then Polmat.

Let us verify our idea by three experimental tests. The first one shows a computing time of solution linear equation with polynomial matrices with using of Polmat and the Polynomial Toolbox, the second one deals with computation of a matrix determinant and compares a computing time of four methods and, finally, third one shows computing times of reduction matrix to Smith form by Polmat, `linalg` and the Polynomial Toolbox.

The random matrices, used in the testing, are generated by `polmat::rand` in MuPAD and `prand` in Matlab. In MuPAD the computing time is measured by a function `time`, in Matlab by `cputime` and in Scilab by `timer`. These functions measure the CPU time that was spent by the current process. Tables bellow contain computing times in the same format as mentioned commands return them (introduced times may be rounded). Tests are performed under operating system Linux with kernel 2.4.25 in MuPAD 2.5.3, in Matlab 6.0.0.88 (R12) with the Polynomial Toolbox 3.0.19 and in Scilab 3.0. Note that tests were running with variable `DIGITS` equal 40 in MuPAD.

7.1 Equation Solvers

As it was said, we solve an equation $\boldsymbol{A}\,\boldsymbol{X} + \boldsymbol{B}\,\boldsymbol{Y} = \boldsymbol{C}$ with using macros

- `polmat::axbyc` in MuPAD and
- `axbyc` in Matlab

in the first test. First, we found general solution equation with polynomial matrices (Tab. 7.1), futher, we take a solution equation with polynomials with a minimal degree of $\boldsymbol{Y}$ (Tab. 7.2).

polynomial degree	1	5	10	1	2	3	3
matrix dimension	1	1	1	2	3	3	5
Polmat	29	122	482	120	844	2056	19 630
Polynomial Toolbox	41	44	47	40	46	47	56

Table 7.1: Benchmark – the computing times in milliseconds: Solver for equation $\boldsymbol{A}\,\boldsymbol{X}+\boldsymbol{B}\,\boldsymbol{Y}=\boldsymbol{C}$, general solution

polynomial degree	3	5	10	25
Polmat	49	83	224	1675
Polynomial Toolbox	86	91	98	128

Table 7.2: Benchmark – the computing times in milliseconds: Solver for equation $\boldsymbol{A}\,\boldsymbol{X}+\boldsymbol{B}\,\boldsymbol{Y}=\boldsymbol{C}$, general solution

7.2 Matrix Determinant

The second test compares five algorithms for a matrix determinant, two symbolic methods and three numeric ones, performed by

- `linalg::det` in MuPAD,
- `polmat::det` in MuPAD,
- `polmat::det` in MuPAD,
- `inv` in the Polynomial Toolbox in Matlab and
- `invr` in Scilab.

The results are contained in Tab. 7.3.

polynomial degree	1	3	3	5	5	10
matrix dimension	2	3	5	5	10	10
MuPAD, `linalg`	2	23	292	656	19 620	*
Polmat, symbolic	26	286	10 448	*	*	*
Polmat, numeric	11	42	92	173	1114	2590
Polynomial Toolbox	2	2	3	4	4	9
Scilab	0	2	0	2	4	8

Table 7.3: Benchmark – the computing times in milliseconds: Determinant of a polynomial matrix n by n of degree d

7.3 Smith Form

Reduction polynomial matrix to Smith form is tested. Three algorithms,

- `linalg::smithForm` in MuPAD,
- `polmat::smithForm` in Polmat,
- `smith` in the Polynomial Toolbox in Matlab,

are compared. The computing times are shown in Tab. 7.4.

polynomial degree	1	3	4	6	2	3
matrix dimension	2	3	3	3	4	4
`linalg`	23	257	502	2590	3614	68310
Polmat	91	910	1554	5100	1198	2944
Polynomial Toolbox	210	785	1040	1465	950	1380

Table 7.4: Benchmark – the computing times in milliseconds: Reduction polynomial matrix to Smith form

Chapter 8

Applications in Control Design

We introduce some basic control problems and their solution by polynomial methods with using Polmat in this chapter.

8.1 Cruise Control System

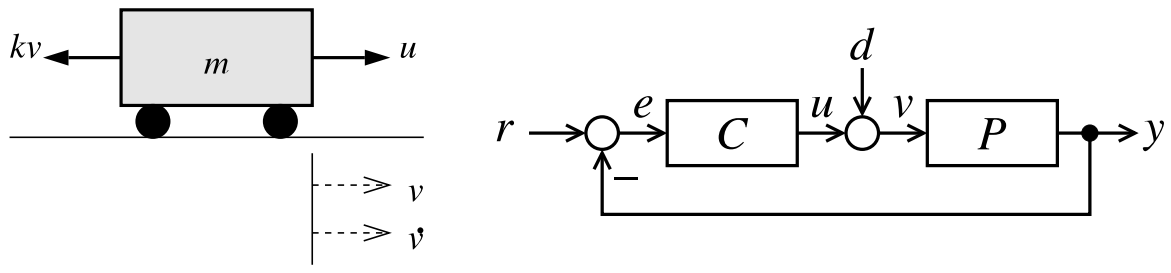
In the first example, consider a simple model of a car [48] of Fig. 8.1a. Neglecting the inertia of the wheels and supposing that friction is what is opposing the motion of the car, the system can be described by

$$m\,\dot{v} + k\,v = u,$$

where u is the force from the engine, v is the velocity and k is the friction coefficient. Let u be input and v output. The corresponding transfer function is

$$P(s) = \frac{1}{m\,s + k}\,. \tag{8.1}$$

In what follows, assume $m > 0$ and $k > 0$ and consider the scheme of Fig. 8.1b.



(a) (b)

Figure 8.1: Cruise control system (a), Feedback control system (b)

8.1.1 Stabilizing Controllers

We shall compute a parametrization of all controllers that internally stabilize the plant (8.1). Set of all such controllers is given by

$$C = \frac{Y - A\,W}{X + B\,W},$$

where X and Y are two proper and stable rational functions satisfying the Bézout equation

$$A\,X + B\,Y = 1$$

and W is a parameter ranging over the set of proper and stable rational functions such that $X + BW \neq 0$. For more details and proof read, e. g., [13, 25].

Obtain a stable fractional representation of the plant (8.1) as follows. Choose a factor $(s-1)$,

$$\frac{B(s)}{A(s)} = \frac{1}{m\,s + k}, \qquad A(s) = \frac{m\,s + k}{s + 1}, \qquad B(s) = \frac{1}{s + 1}$$

and solve the equation

$$\frac{m\,s + k}{s + 1}\,X(s) + \frac{1}{s + 1}\,Y(s) = 1. \tag{8.2}$$

By substitution

$$s = \frac{1 - w}{w},$$

transform (8.2) to the polynomial equation

$$(m + (k - m)\,w)\;\; x(w) + w\,y(w) = 1 \tag{8.3}$$

and solve it by Polmat

```
1  assume(m>0): assume(k>0):
2  A:=(m*s+k)/(s+1):
3  B:=1/(s+1):
4  a:=polmat(normal(subs(A, s=(1-w)/w)), w):
5  b:=polmat(normal(subs(B, s=(1-w)/w)), w):
6  c:=polmat(1, w):
7  t:=0:
8  [x,y]:=polmat::axbyc(a, b, c, w):
```

By assuming $t = 0$ on line 7, get a particular solution of (8.3). Now, perform backsubstitution

$$w = \frac{1}{s + 1}$$

into $x(w)$ and $y(w)$ and obtain solution of (8.2)

```
9  X:=subs(expr(x), w=1/s+1):
10 Y:=subs(expr(y), w=1/s+1):
```

The formula for all controllers that internally stabilize the plant (8.1) reads

$$C(s) = \frac{\frac{m-k}{m} - \frac{m\,s+k}{s+1}\,W(s)}{\frac{1}{m} + \frac{1}{s+1}\,W(s)}. \tag{8.4}$$

For

$$m = 1000\,\mathrm{kg}, \quad k = 50\,\mathrm{N\,s\,m^{-1}}, \tag{8.5}$$

the set of controllers is

$$C(s) = \frac{\frac{19}{20} - \frac{1000\,s+50}{s+1}\,W(s)}{\frac{1}{1000} + \frac{1}{s+1}\,W(s)}.$$

8.1.2 PI Controllers

We computed the set of all controllers that internally stabilize the plant (8.1) in previous subsection. We shall now select all PI controllers contained in the set (8.4).

PI controller has a form

$$C_{\mathrm{PI}}(s) = \frac{k_{\mathrm{P}}\,s + k_{\mathrm{I}}}{s}.$$

Solve the equation

$$C(s) = C_{\mathrm{PI}}(s)$$

for $W(s)$

```
11  C:=(Y-A*W)/(X+B*W):
12  Cpi:=(kP*s+kI)/s:
13  solve(C=Cpi, W):
```

PI controllers corespond to the parameter

$$W(s) = \frac{(m-k-k_{\mathrm{P}})\,s^2 + (m-k-k_{\mathrm{I}}-k_{\mathrm{P}})\,s - k_{\mathrm{I}}}{m^2\,s^2 + m\,(k_{\mathrm{I}}+k_{\mathrm{P}})\,s + m\,k_{\mathrm{I}}}.$$

With the parameters (8.5) $W(s)$ is

$$W(s) = \frac{(950-k_{\mathrm{P}})\,s^2 + (950-k_{\mathrm{I}}-k_{\mathrm{P}})\,s + -k_{\mathrm{I}}}{1000000\,s^2 + (1000\,k_{\mathrm{P}} + 50000)\,s + 1000\,k_{\mathrm{I}}}.$$

8.1.3 Asymptotic Properties

For the system (8.1) find a controller so that

1. the feedback system is internally stable,

2. the final value of y equals 1 when r is a unit step and $d = 0$,
3. the final value of y equals zero when d is a sinusoid of 1 rad s^{-1} and $r = 0$.

First, determine the set of all internally stabilizing controllers. It was done by (8.4). The second step is to compute the input-to-output transfer function and disturbance-to-output transfer function, these read

$$T(s) = B(s)\ (Y(s) - A(s)\,W(s))\,,$$
$$H(s) = B(s)\ (X(s) + B(s)\,W(s))\,,$$

respectively. The point (2) holds if [5]

$$B(0)\ (Y(0) - A(0)\,W(0)) = 1$$

and the point (3) holds if

$$B(j)\ (X(j) + B(j)\,W(j)) = 0.$$

The problem is reduced to solving following three algebraic equations

$$\begin{aligned} W(0) &= -\frac{1}{m} \\ \Re\{W(j)\} &= -\frac{1}{m} \\ \Im\{W(j)\} &= -\frac{1}{m}. \end{aligned} \tag{8.6}$$

Let W be a polynomial in $(s+1)^{-1}$ with enough variable coefficients, so

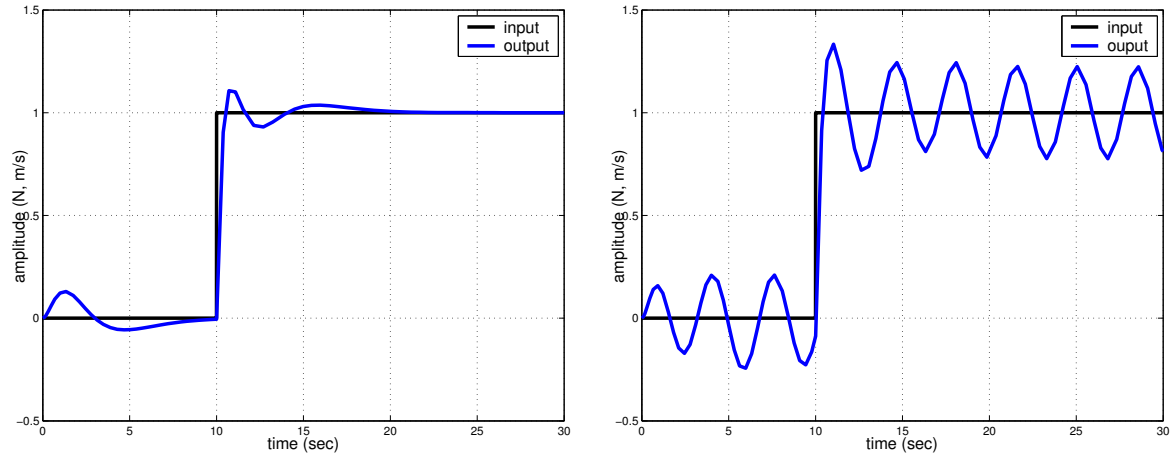
$$W(s) = x_1 + \frac{1}{s+1}\,x_2 + \frac{1}{(s+1)^2}\,x_3.$$

The equations (8.6) can be written as one $\boldsymbol{F}\,\boldsymbol{x} = \boldsymbol{b}$ where

$$\boldsymbol{F} = \begin{pmatrix} 1 & \left.\frac{1}{s+1}\right|_{s=0} & \left.\frac{1}{(s+1)^2}\right|_{s=0} \\ 1 & \left.\Re\left\{\frac{1}{s+1}\right\}\right|_{s=j} & \left.\Re\left\{\frac{1}{(s+1)^2}\right\}\right|_{s=j} \\ 0 & \left.\Im\left\{\frac{1}{s+1}\right\}\right|_{s=j} & \left.\Im\left\{\frac{1}{(s+1)^2}\right\}\right|_{s=j} \end{pmatrix}, \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} W(0) \\ \Re\{W(j)\} \\ \Im\{W(j)\} \end{pmatrix}$$

In MuPAD, type following

```
14  v:=matrix([[1, 1/(s+1), 1/(s+1)^2]]):
15  F:=matrix::stackMatrix(subs(v, s=0),
       map(subs(v, s=I), Re), map(subs(v, s=I),Im)):
16  b:=matrix([op(solve(subs(B*(Y-A*W)=1, s=0), W)),
       Re(op(solve(subs(B*(X+B*W), s=I), W))),
       Im(op(solve(subs(B*(X+B*W), s=I), W)))]):
17  [x1, x2, x3]:=[op(linalg::matlinsolve(F, b))]:
18  Cap:=subs(C, W=x1+1/(s+1)*x2+1/(s+1)^2*x3):
```



(a) (b)

Figure 8.2: Step response of the closed-loop system, disturbance frequency 1 rad s^{-1} (a), 1.8 rad s^{-1} (b)

The controller that achieves internal stability and asymptotic properties is given by

$$C_{\mathrm{ap}}(s) = \frac{(4\,m - k)\,s^3 + 5\,m\,s^2 + (4\,m - k)\,s + m}{s^3 + s}, \tag{8.7}$$

The substituting (8.5) into (8.7) gets

$$C_{\mathrm{ap}}(s) = \frac{3950\,s^3 - 5000\,s^2 - 3950\,s + 1000}{s^3 + s}.$$

Properties of the closed-loop system are best seen in the diagrams. Step response of the close-loop system for disturbance with amplitude 1000 and frequency 1 rad s^{-1} and 1.8 rad s^{-1} is in Fig. 8.2. It is obvious that the output tracks reference and the sinusoid 1 rad s^{-1} is filtered.

8.2 Inverted Pendulum

Consider an inverted pendulum [5, 27, 38] of Fig. 8.3a. Suppose that friction in joint is negligible and all mass is concentrated in the ball. Then describe it by

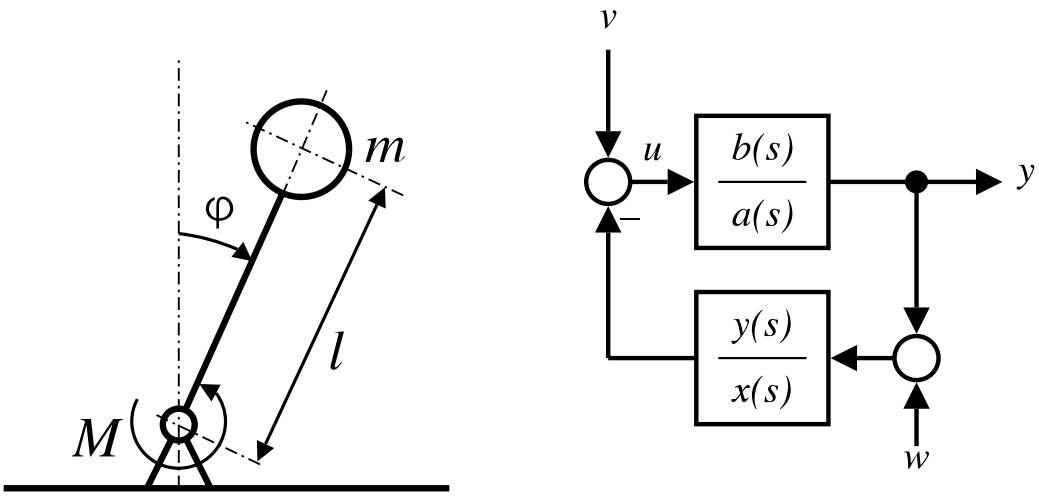
$$m\,l^2\,\frac{\mathrm{d}^2\,\varphi(t)}{\mathrm{d}\,t^2} - m\,g\,l\,\sin\varphi(t) = -M(t) \tag{8.8}$$

where m is mass of the rod, l length of the rod, φ is angle and M is a torque.

Linearize the equation (8.8) about the point $\varphi(t) = 0$ rad and describe an inverted pendulum by transfer function

$$P(s) = \frac{1}{m\,g\,l - m\,l^2\,s^2}$$

where input is the torque M and output is the angle φ. Choose $m = 0.5$ kg and $l = 1$ m and enter this system into MuPAD



(a) (b)

Figure 8.3: Inverted pendulum (a), Feedback control scheme (b)

```
19  P:=1/(m*l*g-m*l^2*s^2):
20  m:=1/2: l:=1: g:=981/100:
21  b:=polmat(numer(P), s):
22  a:=polmat(denom(P), s):
```

With these particular parameters the transfer function of the plant is

$$P(s) = \frac{b(s)}{a(s)} = \frac{200}{981 - 100\,s^2}. \tag{8.9}$$

Evidently, the pendulum up is unstable position and the system (8.9) is not stable.

8.2.1 Pole Placement

Design a controller via pole placement for our system (8.9). Consider the scheme 8.3b and a controller having a form

$$C(s) = \frac{y(s)}{x(s)}.$$

Then the characteristic polynomial of a closed-loop system is

$$c(s) = a(s)\,x(s) + b(s)\,y(s). \tag{8.10}$$

To physically realize a controller, $c(s)$ must satisfy

$$\partial c(s) \geq 2\,\partial a(s) - 1,$$

hence, the characteristic polynomial for the system (8.9) has the form

$$c(s) = (s^2 + 2\,\zeta\,\omega_\mathrm{n}\,s + \omega_\mathrm{n}^2)\,(s + c_1).$$

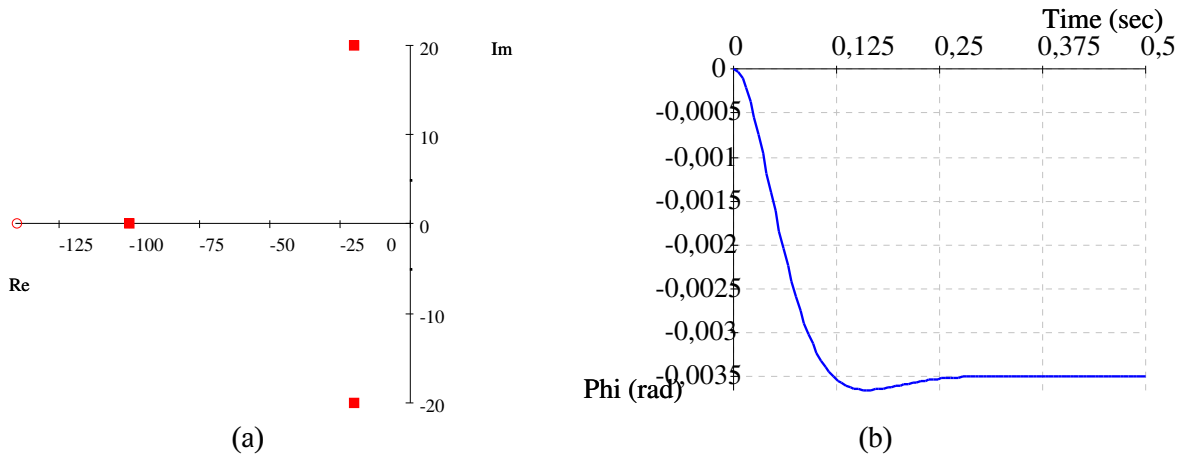


Figure 8.4: Zero and pole map (a) and step response (b) of closed-loop system

Choose

$$\omega_n = 20\sqrt{2}, \qquad \zeta = \frac{\sqrt{2}}{2}, \qquad c_1 = 100,$$

corresponding characteristic polynomial is

$$c(s) = (s^2 + 40\,s + 800)\,(s + 100),$$

and solve (8.10) for $x(s)$ and $y(s)$.

```
23  c:=polmat((s^2+40*s+800)*(s+100), s):
24  [x, y]:=polmat::axbyc(a, b, c, s, "y-minimal"):
25  C:=polmat::tf(y, x):
```

The resulting controller is

$$C(s) = \frac{-480981\,s - 8137340}{200\,s + 28000}.$$

Zero and pole map and step response of closed-loop system is in Fig. 8.4.

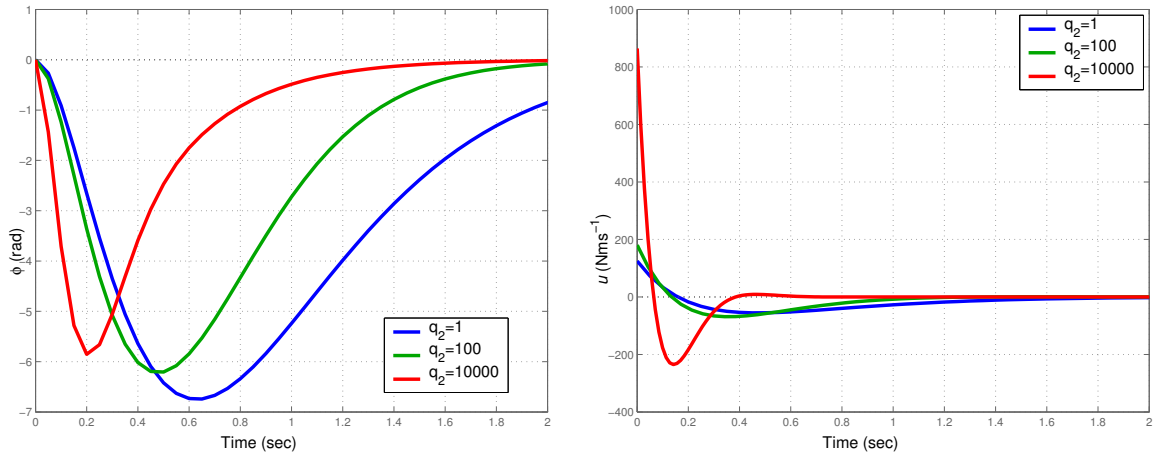
8.2.2 LQG Controller

Our objective is to design a controller

$$C(s) = \frac{y(s)}{x(s)}$$

for the system (8.9) that minimizes the standard quadratic optimization criterion

$$\lim_{t\to\infty} \int_0^t \left\{ u^{\mathrm{T}}(t)\, r_2\, u(t) + y^{\mathrm{T}}(t)\, q_2\, y(t) \right\} \mathrm{d}\,t$$



(a) (b)

Figure 8.5: Impulse response of closed-loop system, output (a), manipulated variable (b) for various weights q_2 (impulse is applied to input w)

where $u(t)$ is the control input and $y(t)$ is the measured output. Let $v(t)$ be the disturbing torque acting on the pendulum with covariance r_1 and $w(t)$ is a noise that adds to measured angular position with covariance q_1, see Fig. 8.3b. Consider the covariances of the input noise and output noise

$$q_1 = 5 \cdot 10^{-6}, \quad r_1 = 5 \cdot 10^{-6}$$

and weights on the output (controlled) signals and the input (control) signal

$$q_2 = 1, \quad r_2 = 1.$$

A controller design consists of solving two spectral factorizations and one linear equation

$$\begin{aligned} a(s)\, r_1\, a^*(s) + b(s)\, q_1\, b^*(s) &= f(s)\, f^*(s) \\ a(s)\, r_2\, a^*(s) + b^*(s)\, q_2\, b(s) &= g(s)\, g^*(s) \\ a(s)\, x(s) + b(s)\, y(s) &= f(s)\, g(s), \end{aligned}$$

where $X^*(s) = X^{\mathrm{T}}(-s)$ and taking the solution with $\partial\, y(s) < \partial\, a(s)$

```
26  q1:=r1:=polmat(5/1000000, s):
27  q2:=r2:=polmat(1, s):
28  f:=polmat::spf(a*r1*polmat::conjugatetranspose(a, s)+
       b*q1*polmat::conjugatetranspose(b, s), s)[1]:
29  g:=polmat::spf(a*r2*polmat::conjugatetranspose(a, s)+
       polmat::conjugatetranspose(b, s)*q2*b, s)[1]:
30  c:=polmat(f*g, s):
31  [x, y]:=polmat::axbyc(a, b, c, s, "degy<dega"):
32  subs([x, y], t=0):
33  C:=simplify(factor(1/polmat::tf(op(%)))):
```

The controller is

$$C(s) = \frac{1944722\sqrt{2} + 1962\sqrt{2004722} + s\sqrt{\sqrt{1002361} + 981}\left(20\sqrt{1002361} + 19620\right)}{-294300\sqrt{2} - 400\sqrt{2004722} \quad - 10000\,s^2\sqrt{2} - 4000\,s\sqrt{\sqrt{1002361} + 981}},$$

numerically

$$C(s) = \frac{1764999.42\,s + 5528212.82}{-178086.71\,s - 14142.14\,s^2 - 982555.87}.$$

Fig. 8.5 shows impulse responses for various q_2.

8.3 Pendulum Discrete-Time Model

This example shows usage of Polmat in discrete-time control theory. Consider a system of Fig. 8.3a. Discrete-time model, corresponding to pendulum up position, can be written as

$$P(d) = \frac{b(d)}{a(d)} = \frac{d^2 + 2\,d + 1}{8\,d^2 + 24\,d + 8} \tag{8.11}$$

where a sample time was chossen $T_s = 1$. In following, consider scheme of Fig. 8.6.

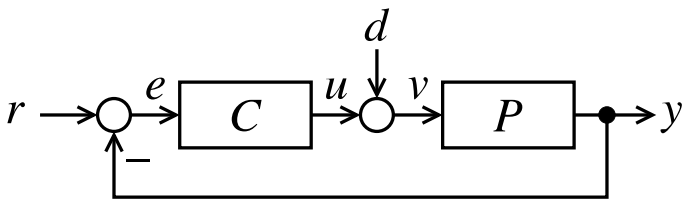


Figure 8.6: Feedback control scheme

8.3.1 Stabilizing Controllers

At first, compute set of all stabilizing controllers for the system (8.11). It is defined as

$$C(d) = \frac{y - a\,w}{x + b\,w},$$

where x and y satisfy

$$a\,x + b\,y = 1$$

and w is an arbitrary stable and proper rational function such that $x + b\,w \neq 0$.

Solve this task in MuPAD by

```
34 P:=(d^2+2*d+1)/(8*d^2+24*d+8):
35 a:=polmat(denom(P)):
36 b:=polmat(numer(P)):
37 [x, y]:=polmat::axbyc(a, b, 1, d):
38 [x, y]:=subs([x, y], t=0):
39 C:=polmat::tf(y-a*w, x+b*w):
```

All stabilizing controllers are given by

$$C(d) = \frac{8\,d + 24 - (64\,d^2 + 192\,d + 64)\,w}{-d - 2 + (8\,d^2 + 16\,d + 8)\,w}.$$

8.3.2 Asymptotic Reference Tracking

Output tracks reference of type step if denominator of open-loop system contains $(1 - d)$. Plant (8.11) has not this property, so a controller must have it. Solve equation

$$x + b\,w = -d - 2 + (8\,d^2 + 16\,d + 8)\,w = (1 - d)\,\widehat{w}$$

for w. Type

```
40 solve(denom(C)=(1-d)*w_, w):
41 Crt:=subs(C, w=op(%)):
```

and get

$$C_{\mathrm{rt}}(d) = \frac{-8 - (8\,d^3 + 16\,d^2 - 16\,d - 8)\,\widehat{w}}{(d^3 + d^2 - d - 1)\,\widehat{w}}, \tag{8.12}$$

where $\widehat{w}$ is an arbitrary stable and proper rational function such that

$$(d^3 + d^2 - d - 1)\,\widehat{w} \neq 0. \tag{8.13}$$

Step response of closed-loop system for $\widehat{w} = 0.1$ and $\widehat{w} = 0.0001$ is in Fig. 8.7.

8.3.3 LQ Controller

Find an LQ controller contained in the set (8.12). Disturbance-to-output transfer function is given by

$$G(d) = \frac{P}{1 + P\,C_{\mathrm{rt}}} = p - q\,\widehat{w} = 0 - (d^3 + 2\,d^2 - 2\,d - 1)\,\widehat{w}. \tag{8.14}$$

$\mathcal{H}_2$ norm of (8.14) is

$$\|G(d)\| = \|p - q\,\widehat{w}\| = \| - (d^3 + 2\,d^2 - 2\,d - 1)\,\widehat{w}\|,$$

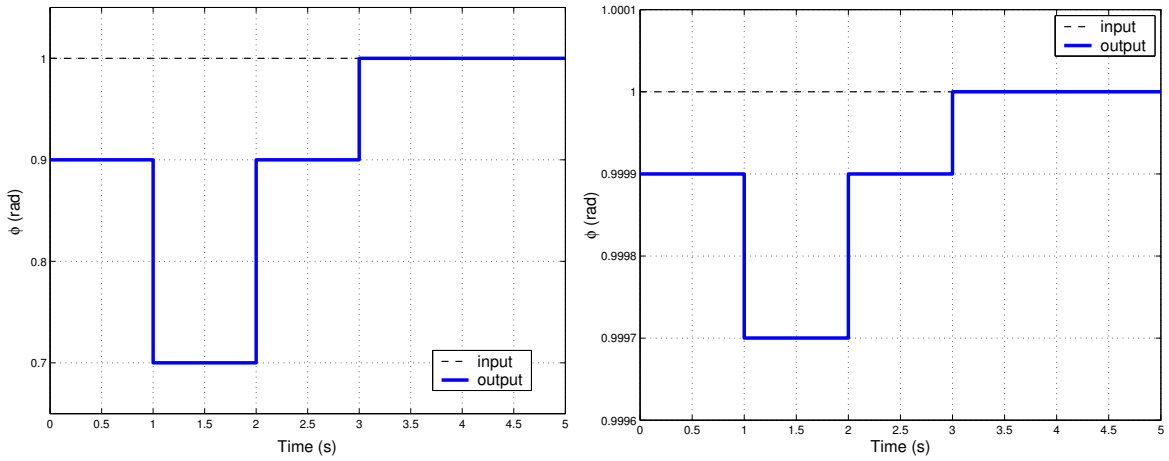
clearly, it is zero for

$$\widehat{w} = 0. \tag{8.15}$$

With using of Polmat you can type

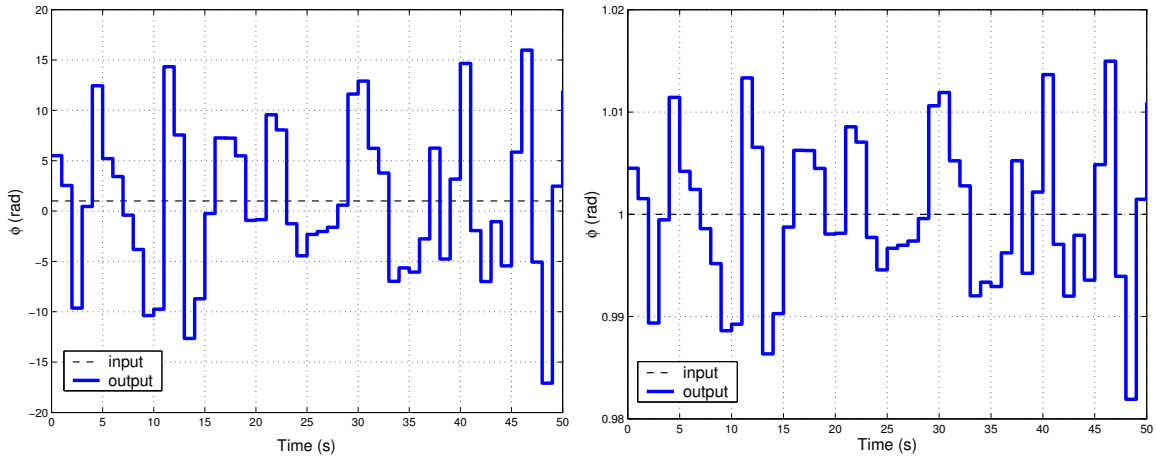
```
42  G:=collect(normal(P/(1+P*Crt)), w_):
43  wh2:=-polmat::coeff(G, w_, 0)/polmat::coeff(G, w_, 1):
```

Solution (8.15) does not satisfy (8.13). Optimal solution of LQ controller problem was not found. LQ controller tracking reference of type step does not exist. Step response of closed-loop system for value near to optimal ($\widehat{w} = 0.0001$) is in Fig. 8.7b. Fig. 8.8 shows step responses with disturbance d with zero mean and variance equaled 100000.



(a) (b)

Figure 8.7: Asymptotic reference tracking, step response of closed-loop system for $\widehat{w} = 0.1$ (a), $\widehat{w} = 0.0001$ (b)



(a) (b)

Figure 8.8: Asymptotic reference tracking, step response of closed-loop system for $\widehat{w} = 0.1$ (a), $\widehat{w} = 0.0001$ (b) disturbed by d with zero mean and variance equaled 100000

8.4 Robust Control

This example shows using of Polmat in robust control theory. Consider a primitive mechanical system given by transfer function [11]

$$\frac{\Theta(s)}{T(s)} = \frac{10\,d\,s + 10\,k}{s^2(s^2 + 11\,d\,s + 11\,k)} = \frac{b(s)}{a(s)}$$

where Θ denotes an angle, T is an applied moment and k and d are undetermined parameters being found in the intervals

$$0.09 \le k \le 0.4, \quad 0.004 \le d \le 0.04. \tag{8.16}$$

Design a controller and find out *robust stability* [5, 7, 11].

At first, choose any d and k agreeing with (8.16) and enter the nominal model

```
[dmin, dmax]:=[4/1000, 4/100]:
[kmin, kmax]:=[9/10, 4/10]:
d:=(dmin+dmax)/2:
k:=(kmin+kmax)/2:
a:=polmat(s^2*(s^2+11*d*s+11*k)):
b:=polmat(10*d*s+10*k):
```

The nominal model is

$$\frac{b(s)}{a(s)} = \frac{110\,s + 3250}{500\,s^4 + 121\,s^3 + 3575\,s^2}.$$

Choose the characteristic polynomial of the closed loop and design a controller via pole placement

```
c:=polmat((s+5)^5*((s+1/2+1/4*I)*(s+1/2-1/4*I))):
[x,y]:=polmat::axbyc(a, b, c, s, "y-minimal"):
```

The controller

$$C(s) = \frac{12963581441757\,s^3 - 10637501072975\,s^2 + 22346289062500\,s + 5363769531250}{5701250000\,s^3 + 919592797500\,s^2 + 9351194996130\,s + 42140725407250}.$$

is obtained.

Now, perform the robust stability test

```
delete d, k
a:=polmat(s^2*(s^2+11*d*s+11*k), s):
b:=polmat(10*d*s+10*k, s):
cl:=collect(a*x+b*y, [k, d]):
pd:=coeff(cl, d, 1):
pk:=coeff(cl, k, 1):
p0:=cl-d*pd-k*pk:
polmat::ptopplot(p0, [pd, pk], [[dmin, dmax], [kmin, kmax]],
x*I/100 $ x=0..150)
```

The last command plots a value set of polytope of polynomials (Fig. 8.9). It is obvious that zero is excluded and in accordance with *Zero Exclusion Theorem* [11] the closed loop is robustly stable.

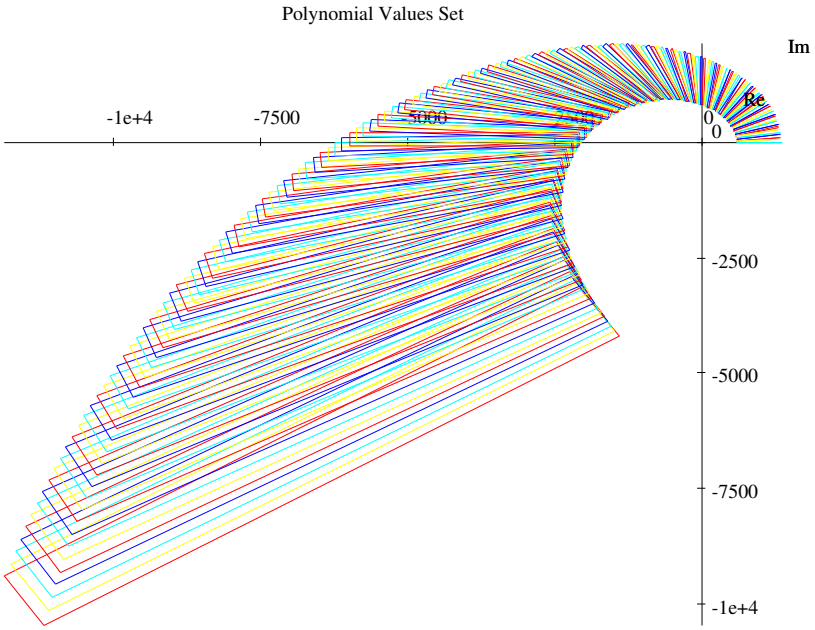


Figure 8.9: Value set of polytope of polynomials

Chapter 9

Conclusions

It was the aim of this diploma thesis to give a package of functions for symbolic computation with polynomials and polynomial matrices for the computer algebraic system MuPAD. The library was named Polmat. MuPAD with Polmat provide a useful alternative to the few existing numerical libraries for polynomial matrices. Polmat is available for free download [1] and responses from control community as well as MuPAD development team have been stated.

MuPAD was found efficient and very capable mathematical tool and great package for symbolic computation. Polmat makes possible using MuPAD in control theory by a series of new implemented functions, including solvers for linear equations with polynomial matrices and spectral factorization.

Implemented algorithms do not make ambitions to perform calculations faster than similarly oriented packages, based on numerical methods. In fact, benchmarks in Sec. 7 show that computing times needed to return result are in majority longer in Polmat than in other well-known packages for polynomial matrices computation. Solving linear equations with polynomials or polynomial matrices gets longer time with using of Polmat than the Polynomial Toolbox for Matlab. While computing time increases quickly with matrix dimension and polynomial degree in Polmat, it seems to be constant in Polynomial Toolbox. There have been implemented two methods for computing a matrix determinant, symbolic and numeric one. The symbolic algorithm is the slowest of all tested ones. The numeric method is faster than `linalg::det` but much slower then functions of numerical packages. While computing time of reduction matrix to Smith form by `linalg` library increases quickly with a matrix dimension, it increases rather with polynomial degree in Polmat.

There is one relevant problem with using Polmat and MuPAD which might bring some difficulties. Entry of polynomial matrix or 1 by 1 polynomial matrix as it is defined in Polmat (see Sec. 5) is not compatible with domain `DOM_POLY` which is used for scalar polynomial. Domain `DOM_POLY` is used because it is provided by the MuPAD kernel and computing with scalar polynomials is so more efficient than with another domains. Similar domain for matrix of polynomials is lacked. This makes problems with performing elementary operations on polynomial matrix when a matrix row or column is multiplied by scalar polynomial. A Polmat user might meet this problem, for instance, if solver an equation with polynomial matrices returns a 1 by 1 matrix whose entry becomes to one of several scalar input arguments of another function. I hope this problem will be fixed during my cooperation with MuPAD team.

Only one of the most important algorithms useful in control theory has not been implemented. It is polynomial matrix spectral factorization for the discrete-time case. It is a future extension.

Bibliography

[1] AUGUSTA, P. – HURÁK, Z. Polmat – polynomials and polynomial matrices for MuPAD [online]. ⟨`http://www.polmat.wz.cz`⟩, 2004. [cite 2004-07-11].

[2] AUGUSTA, P. – HURÁK, Z. *Polmat user's guide. Reference manual*, December 2004.

[3] CHIASSON, J. et al. Control of a multilevel converter using resultant theory. *IEEE Transactions on Control System Technology*, vol. 11:345–354, May 2003.

[4] DEMLOVÁ, M. – NAGY, J. *Algebra*. Praha: SNTL, 1985.

[5] DOYLE, J. C. – FRANCIS, B. A. – TANNENBAUM, A. R. *Feedback Control Theory*. New York: Macmillan, 1992.

[6] HALMO, L. Polpack++: C++ library for computing with polynomial matrices [online]. ⟨`http://polpackplusplus.sourceforge.net`⟩, February 2004.

[7] HAVLENA, V. – ŠTECHA, J. *Moderní teorie řízení*. Praha: Vydavatelství ČVUT, 2st edition, 2000.

[8] HENRION, D. *Reliable algorithms for polynomial matrices*. PhD thesis, Institute of Information Theory and Automation, ASCR. Prague, 1998.

[9] HENRION, D. – PRIEUR, C. – TLIBA, S. Improving conditioning of polynomial pole placement problems with application to low-order controller design for a flexible beam. *LAAS-CNRS Research Report No. 04163*, February 2004.

[10] HENRION, D. – ŠEBEK, M. An algorithm for polynomial matrix factor extraction. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1875–1880, Phoenix, Arizona, December 1999.

[11] HROMČÍK, M. – HURÁK, Z. – ŠEBEK, M. Robustní řízení [online]. ⟨`http://dce.felk.cvut.cz/ror`⟩, 2004. [cite 2004-11-11].

[12] HROMČÍK, M. – ŠEBEK, M. New algorithm for polynomial matrix determinant based on FFT. In *European Control Conference ECC'99*, Karlsruhe, Germany, September 1999.

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

[14] JEŽEK, J. New algorithm for minimal solution of linear polynomial equations. *Kybernetika*, vol. 18, no. 6:505–516, 1982.

[15] JEŽEK, J. Conjugate and symmetric polynomial equation – I : Continue-time systems. *Kybernetika*, vol. 19, no. 2:121–130, 1983.

[16] JEŽEK, J. Conjugate and symmetric polynomial equations – II : Discrete-time systems. *Kybernetika*, vol. 19, no. 3:196–211, 1983.

[17] JEŽEK, J. – KUČERA, V. Efficient algorithm for matrix spectral factorization. *Automatica*, vol. 21, no. 6:663–669, 1985.

[18] JOHN, J. *Systémy a řízení*. Praha: Vydavatelství ČVUT, 1st edition, 1999. ISBN 80-01-01474-6.

[19] KAILATH, T. *Linear Systems*. New York : Prentice Hall, 1980.

[20] KRAJNÍK, E. *Maticový počet*. Praha: Vydavatelství ČVUT, 1st edition, 2000. ISBN 80-01-01723-0.

[21] KUJAN, P. Efektivní výpočty s polynomiálními maticemi v systému Mathematica s aplikacemi v řízení. Master's thesis, Czech Technical University in Prague, Department of Control Engineering. Prague, 2004.

[22] KUJAN, P. – HROMČÍK, M. – ŠEBEK, M. New package for effective polynomial computation in Mathematica. In *The 11th Mediterranean Conference on Control and Automation (MED'03)*, July 2003. ISBN 960-87706-0-2.

[23] KUČERA, V. *Algebraická teorie diskrétního lineárního řízení*. Praha: Academia, 1978.

[24] KUČERA, V. *Discrete Linear Control*. John Wiley and Sons, 1979.

[25] KUČERA, V. Parametrization of stabilizing controllers with applications. *Advances in Automatic Control,* Voicu, M., editor, pages 173–192, 2003.

[26] KWAKERNAAK, H. – ŠEBEK, M. Polynomial J-spectral factorization. *IEEE Transactions on Automatic Control*, vol. 39, no. 2:315–328, 1994.

[27] ЛЕОНОВ, Г. А. *Лекции по курсу теория управления I*. Санкт-Петербургский государственный университет, 2004.

[28] LUKÁČ, S. Implementácia IKT do vyučovania matematiky. In *3. celoštátna konferencia Infovek 2002*, Október 2002.

[29] METZNER, T. *The MuPAD-Scilab link*. ⟨`http://www.additive-net.de/ftp/win32/software/mupad/MuPADScilab_Doku.pdf`⟩, 2003.

[30] MULDERS, T. – STORJOHANN, A. On lattice reduction for polynomial matrices. *Journal of Symbolic Computation*, no. 35:377–401, 2003.

[31] OEVEL, W. – GERHARD, J. *MuPAD 2.5 Quick Reference*. 2002.

[32] OEVEL, W. et al. *The MuPAD Tutorial*. Springer, 2000.

[33] PADĚRA, M. Numerical algorithms for polynomial matrices in Java. Master's thesis, Czech Technical University in Prague, Department of Control Engineering. Prague, 2004.

[34] POSTEL, F. The linear algebra package "linalg". *Automath Technical Report*, no. 9, 1998.

[35] QIU, L. What can Routh table offer in addition to stability? In *4th IFAC Symposium on Robust Control design*, 2003.

[36] ROUBAL, J. et al. *Moderní teorie řízení – Cvičení*. Praha: Vydavatelství ČVUT, 1st edition, 2005.

[37] ŠTECHA, J. *Diskrétní řídicí systémy*. Praha: České vysoké učení technické v Praze, 1st edition, 1985.

[38] ŠTECHA, J. – HAVLENA, V. *Teorie dynamických systémů*. Praha: Vydavatelství ČVUT, 2nd edition, 2002. ISBN 80-01-01971-3.

[39] ŠTEFKO, P. C library for computing with polynomial matrices for the TI-89/92 programmable calculators [online]. ⟨`http://ptoolti89.wz.cz`⟩, February 2004.

[40] VOSTRÝ, Z. New algorithm for polynomial spectral factorization with quadratic convergence I. *Kybernetika*, vol. 11, no. 6:415–422, 1975.

[41] VOSTRÝ, Z. New algorithm for polynomial spectral factorization with quadratic convergence II. *Kybernetika*, vol. 12, no. 4:248–259, 1976.

[42] ZIPPEL, R. *Effective polynomial computation*. Boston : Kluwer Academic Publishers, 1993. ISBN 0-7923-9375-9.

[43] Maple home page. Maplesoft, a division of Waterloo Maple, Inc. [online]. ⟨`http://www.maplesoft.com`⟩, 2004. [cite 2004-11-04].

[44] The polynomial toolboox 2.0 – polynomial equations, polynomial matrices. PolyX, Ltd. [online]. ⟨`http://www.polyx.cz`⟩, 2000. [cite 2004-04-26].

[45] MuPAD. SciFace [online]. ⟨`http://www.mupad.com`⟩, 2004. [cite 2004-08-07].

[46] Scilab. Scilab Group [online]. ⟨`http://scilabsoft.inria.fr`⟩, 2004. [cite 2004-08-07].

[47] The Mathematica home page. Wolfram Research, Inc. [online]. ⟨`http://www.mathematica.com`⟩, 2004. [cite 2004-11-04].

[48] Control tutorial for Matlab [online]. ⟨`http://www.engin.umich.edu/group/ctm`⟩. [cite 2004-08-30].

[49] Wikipedia – the free encyclopedia [online]. ⟨`http://en.wikipedia.org`⟩, 2004. [cite 2003-11-10].

List of Symbols

a	polynomial, number
$\boldsymbol{A}$, $\boldsymbol{A}$(s)	matrix, polynomial matrix
∂a	degree of a polynomial
$\partial \boldsymbol{A}$	degree of a polynomial matrix
$\boldsymbol{I}, \boldsymbol{I}_n$	identity matrix, identity matrix of the dimension $n \times n$
$\boldsymbol{A}_{l,m}$, $\boldsymbol{A}(s)_{l,m}$	matrix of the dimension $l \times m$
A, $A(s)$	transfer function or rational function
$\bar{a}$	complex conjugate of the polynomial
$\tilde{a}$	reciprocal of the polynomial
$(.,.)$	greatest common divisor
$\boldsymbol{A}^{-1}$	inverse of the matrix
adj$\boldsymbol{A}$	adjoint of the matrix
det$\boldsymbol{A}$	determinant of the matrix
diag$_{l,m}$[.]	diagonal matrix
$\|.\|$	norm
$\|.\|_\infty$	$\mathcal{H}_\infty$ norm

List of Figures

List of Tables

Index