Some Algorithms used in the Polynomial Toolbox for Matlab*

Didier Henrion[1,2]
Martin Hromčík[3]
Michael Šebek[3]

December 1, 2000

Abstract

We describe the main lines of some algorithms used in the Polynomial Toolbox for Matlab, namely triangularization of polynomial matrices with numerically stable operations, determinant evaluation and spectral factorization with DFT, and LMI robust stability analysis of polynomial matrix polytopes.

1 Introduction

Since the seventies and the seminal work of Kučera [Kučera, 1979], a sheer volume of literature has been devoted to the polynomial (or algebraic) approach to system control, lending new insights into several fundamental analysis and design issues. In broad terms, the line of attack of polynomial methods consists of establishing the equivalence between solving control problems and associated matrix polynomial equations. It has been shown that almost all control problems at hand are amenable to this kind of treatment, and significant results have been achieved that illustrate the power and elegance of the approach. As a matter of fact, matrix equations over the ring of polynomials play a key role in the realm of control theory.

[1]Laboratoire d'Analyse et d'Architecture des Systèmes, Centre National de la Recherche Scientifique, 7 Avenue du Colonel Roche, 31 077 Toulouse, cedex 4, France. FAX: +33 5 61 33 69 69. E-mail: `henrion@laas.fr`

[2]Also with the Institute of Information Theory and Automation, Czech Academy of Sciences, Pod vodárenskou věží 4, 182 08 Praha, Czech Republic.

[3]Center of Applied Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 166 27 Praha 6, Czech Republic.

*This work was supported by the Ministry of Education of the Czech Republic under contract No. LN00B096.

During the last two decades, major breakthroughs in numerical linear algebra and the always increasing availability of more powerful computational tools have mostly been exploited by control engineers pursing the state-space approach to system control. Perhaps due to historical reasons, numerically reliable and efficient algorithms were mainly developed in the scope of the state-space approach, in contrast with the polynomial approach, which was rather theoretically oriented or aimed at educational purposes. Actually, only very recently it has been noticed that powerful and tractable numerical routines may also be devised to deal with control problems involving polynomial matrices. This is far from being surprising since both state-space and polynomial approaches are known to be closely related.

A Matlab[1] package has recently been developed to solve problems in control but also signal processing with the polynomial approach. As a part of the MathWorks partner program, the package is commercialized by the small Czech company PolyX [PolyX, 2000]. Macros in the toolbox systematically rely on well-spread numerical procedures, the main emphasis being numerical reliability and efficiency. The objective of this paper is to give a brief overview of some algorithms used in the Polynomial Toolbox for Matlab.

We first describe in section 2 how triangularization of polynomial matrices can be performed in a numerically reliable way. In sections 3 and 4 the discrete Fourier transform (DFT) is used to perform two basic tasks over polynomials, namely polynomial matrix determinant evaluation and spectral factorization. Finally, in section 5 we show how convex optimization over linear matrix inequalities (LMIs, see [Boyd et al., 1994]) can be used to assess robust stability of a polytope of matrix polynomials.

2 Triangularization

A common theme when performing computations involving polynomials is transforming a given polynomial matrix into a simpler equivalent form while preserving key properties of the original matrix. The triangularization over the ring of polynomials is a fundamental construction in this regard. Reduction of polynomial matrices into triangular form has crucial relevance in linear system theory. Applications include solving Diophantine equations, extracting the greatest common divisor of polynomial matrices, computing the determinant of a polynomial matrix [Kučera, 1979], as well as obtaining a triangular basis for modules over polynomial domains.

Triangularization of a polynomial matrix $A(z)$ over the ring of polynomials is equivalent to the problem of finding a unimodular polynomial matrix $U(z)$ and a lower-triangular polynomial matrix $T(z)$ such that

$$A(z)U(z) = T(z).$$

In order to devise an algorithm for finding $U(z)$ and $T(z)$, it is first shown in [Henrion and Šebek, 1999] that triangularizing $A(z)$ over the field of rational functions amounts to triangularizing a

[1] Matlab is a trademark of The MathWorks, Inc.

row permuted Sylvester matrix of $A(z)$ over the field of reals, provided one knows the degree of the reduction matrix. Since this degree is usually not known beforehand, we show that it is sufficient to have an upper-bound. The next step is then to convert the problem of triangularization over the field of rational functions to triangularization over the ring of polynomials. This is achieved through the algebraic notion of degree minimization over ideals, see [Henrion and Šebek, 1999] for details.

Since there exists efficient tools of numerical linear algebra such as the QR decomposition by Householder transformations to perform triangularization over the field of reals, our triangularization algorithm for polynomial matrices is numerically reliable as well. The advantage of using this new polynomial triangularization technique over using the standard Euclidean division algorithm may be compared to the advantage of using orthogonal reductions over using the standard Gaussian elimination algorithm when solving linear systems of equations.

Numerical tests have been carried out to compare the performances of our algorithm with other triangularization methods:

Algorithm 2.1 – Sylvester matrix approach presented here [Henrion and Šebek, 1999]

Algorithm 2.2 – Gaussian elimination algorithm described in [Labhalla et al., 1996]

Algorithm 2.3 – Standard elementary polynomial operations [Kučera, 1979]

Algorithm 2.4 – Maple[2] built-in symbolic routine `linalg[hermite]`.

The routines have been tested on square polynomial matrices of dimension n, degree d and random integer coefficients with absolute value strictly less than 10. Excepting algorithm 2.4, the algorithms have been implemented in Matlab 5.1 on a Sun SparcStation 4. The criterion that has been used to decide whether the triangularization has properly been achieved is twofold:

1. All coefficients in the strictly upper triangular part of the triangular form $T(z) = A(z)U(z)$ must have absolute value less than 10^{-8}.
2. The inequalities
$$0.9 < \frac{|\det U(z)|}{|\det U(0)|} < 1.1 \tag{1}$$
must hold for any $z \in [-1, +1]$. This is aimed at testing unimodularity of $U(z)$ without computing $\det U(z)$, a numerically difficult task for high values of n and d.

Average execution times in seconds are reported in table 2.1. A star ($\star$) means that the triangularization failed because the above criteria were not verified. A double star ($\star\star$) means that the execution time exceeded 1500 seconds.

[2] Maple is a trademark of Waterloo Maple Software.

		Algorithm			
d	n	2.1	2.2	2.3	2.4
3	3	0.21	0.62	16	0.65
5	3	0.51	1.1	26	2.7
7	3	0.87	1.6	39	12
9	3	1.5	2.3	54	30
3	4	0.46	1.1	29	8.8
5	4	2.0	2.5	59	111
7	4	4.1	3.7	$\star$	603
9	4	7.8	5.5	$\star$	$\star\star$
3	5	2.1	2.3	$\star$	109
5	5	7.3	4.6	$\star$	$\star\star$
7	5	14.4	8.0	$\star$	$\star\star$
9	5	29	12	$\star$	$\star\star$
3	9	48	17	$\star$	$\star\star$
5	9	220	$\star$	$\star$	$\star\star$
7	9	580	$\star$	$\star$	$\star\star$
9	9	1200	$\star$	$\star$	$\star\star$

Table 2.1: Comparative execution times in seconds.

We can observe that algorithm 2.3 is the less reliable triangularization method. Even for relatively low values of d and n, algorithm 2.3 returns a result that is generally extremely far from the correct result. Algorithm 2.4 is based on symbolic computation. This approach is known to be subject to intermediate expression swell and turns out to be impractical even for medium-sized problems. For low values of n, algorithm 2.1 is faster than algorithm 2.2. The trend is reversed when n is higher. In the latter case, algorithm 2.1 seems to be more reliable than algorithm 2.2, at the price of an increased computational load.

3 Polynomial Matrix Determinant

In this section we show how interpolation can be used to perform fast and accurate evaluation of the determinant of a polynomial matrix, an operation frequently occurring when manipulating rational transfer functions.

Interpolation is a useful technique to handle polynomials and polynomial matrices. To avoid numerical difficulties due to possible bad conditioning of Vandermonde matrices, interpolation can be performed with the discrete Fourier transform (DFT). For a vector $\boldsymbol{p} = [p_0, p_1, \ldots, p_N]$, its direct DFT is defined as the vector $\boldsymbol{y} = [y_0, y_1, \ldots, y_N]$, where

$$y_k = \sum_{i=0}^{N} p_i e^{-j\frac{2\pi k}{N+1}i} \tag{2}$$

and $j = \sqrt{-1}$. Given the vector $\boldsymbol{y} = [y_0, y_1, \ldots, y_N]$, the original vector $\boldsymbol{p}$ can be recovered by inverse DFT as $\boldsymbol{p} = [p_0, p_1, \ldots, p_N]$, where

$$p_i = \frac{1}{N+1} \sum_{k=0}^{N} y_k e^{j \frac{2\pi i}{N+1} k} \tag{3}$$

Now, if s_k denotes $e^{-j\frac{2\pi k}{N+1}}$ and $p(s) = p_0 + p_1 s + \cdots + p_N s^N$ is a polynomial, then the direct DFT of its coefficient vector $\boldsymbol{p} = [p_0, p_1, \ldots, p_N]$ is the vector of values $\boldsymbol{y} = [y_0, y_1, \ldots, y_N]$, where $y_k = \sum_{i=0}^{N} p_i s_k^i$ is the polynomial $p(s)$ evaluated at the point s_k.

Being a standard tool, the FFT algorithm is available as a built-in function of many computing packages, including Matlab. Based on these ideas, a numerical method for evaluating the determinant of a polynomial matrix can be easily developed:

Input: Square polynomial matrix $P(s)$ of size n and degree N.

Output: Polynomial $p(s)$ - the determinant of $P(s)$.

Step 1 Compute the upper bound for the degree of the determinant:

$$N_{det} = \min\{\sum_{i=1}^{n} \deg_{c_i}(P(s)), \sum_{i=1}^{n} \deg_{r_i}(P(s))\} \tag{4}$$

where $\deg_{c_i}(P(s))$ and $\deg_{r_i}(P(s))$ are the i-th column and row degrees of $P(s)$ respectively.

Step 2 Using the FFT algorithm, perform direct DFT at $N_{det}+1$ points on the set $\{P_0, P_1, \ldots, P_N\}$ of coefficient matrices of $P(s)$ and obtain the set $\{Y_k | k = 0, 1, \ldots, N_{det}\}$.

Step 3 Compute the vector $\boldsymbol{z} = [z_0, z_1, \ldots, z_{N_{det}}]$, where $z_k = \det(Y_k), k = 0, 1, \ldots, N_{det}$.

Step 4 Perform inverse DFT on $\boldsymbol{z}$ using the FFT algorithm to obtain the coefficient vector $\boldsymbol{p} = [p_0, p_1, \ldots, p_{N_{det}}]$ of the determinant $p(s) = p_0 + p_1 s + \cdots + p_{N_{det}} s^{N_{det}}$.

In addition to the above algorithm, the computation of the rank, inverse and 2-D determinant of a polynomial matrix have also been implemented in the Polynomial Toolbox. 2-D determinant computation is particularly useful when working with polynomials subject to one uncertain parameter entering polynomially in the coefficients.

According to our experience, numerical algorithms based on DFT feature beneficial computational efficiency and reliability in comparison with other existing methods. To illustrate this, we present the results of some experimental testing in the following tables.

Table 3.1 contains average execution times in seconds for the following algorithms:

Algorithm 3.1 DFT based method proposed here

Algorithm 3.2 generalized-eigenvalues method

Algorithm 3.3 general interpolation method with purely imaginary equidistantly spaced interpolating points

Algorithm 3.4 modified Fadeev-Leverrier method

Algorithm 3.5 symbolic computation (performed by macro `det` of the Symbolic Toolbox for Matlab)

Square test matrices of size n and degree N have random decimal 15-digits coefficients with absolute values less or equal to one. A star ($*$) means that the execution time exceeds 1000 seconds.

		Algorithm				
n	N	3.1	3.2	3.3	3.4	3.5
5	5	0.05	1.1	0.75	1.8	25
10	10	0.5	10	7.5	18	$*$
20	10	5.7	78	33	240	$*$

Table 3.1: Comparative execution times in seconds.

Then we compared the accuracy of the methods in the particular case of unimodular polynomial matrix (i.e. matrix with unity determinant). The entries of table 3.2 are residues defined as the maximum of absolute values of the coefficients of polynomial $p(s) - 1$, where $p(s)$ stands for the computed determinant. Double stars ($**$) stand for inaccurate results with residues greater than 10^2.

		Algorithm				
n	N	3.1	3.2	3.3	3.4	3.5
5	5	10^{-14}	10^{-13}	10^{-8}	10^{-12}	0
10	10	10^{-9}	10^{-3}	$**$	0	0
20	10	10^{-2}	$**$	$**$	$**$	0

Table 3.2: Accuracy tests for unimodular matrices.

We observe that the methods 3.2, 3.3 and 3.4 become inaccurate for large input matrices. The symbolic computation 3.5 is accurate, but long execution times make this approach practically unusable. As a result, our new algorithm 3.1 remains the only reliable and usable method for large matrices of higher degrees.

4 Spectral Factorization

Besides its relationship to polynomial interpolation, DFT is also closely connected to the Z-transform theory which can be used to solve the spectral factorization of a discrete-time symmetric two-sided polynomial.

A two-sided polynomial

$$m(z) = \sum_{i=-d}^{d} m_i z^i,$$

where m_i are complex numbers, is said to be discrete-time symmetric if it equals its adjoint $m^{\sim}(z)$ defined as $m^{\sim}(z) = \overline{m(\bar{z}^{-1})} = \sum_{i=-d}^{d} \overline{m}_i z^{-i}$. Here the bar stands for complex conjugate. Clearly, $m(z)$ is symmetric if $m_{-i} = \overline{m}_i$. It is also evident that the symmetry makes $m(z)$ real for $|z| = 1$. In addition, we require $m(z)$ being positive here. The spectral factor of such $m(z)$ is then the one-sided polynomial $x(z) = x_0 + x_1 z^{-1} + \cdots + x_d z^{-d}$ such that

$$m(z) = x(z) x^{\sim}(z) \tag{5}$$

holds and $x(z)$ is Schur stable, that is, has all its roots z_i inside the unit circle in the complex plane. Up to the sign, the problem has a unique solution.

In order to solve the above equation, we can logarithmize it. As $m(z)$ is analytic and nonzero in $1 - \varepsilon < |z| < 1 + \varepsilon$ while $x(z)$ is analytic and nonzero in $1 - \varepsilon < |z|$ including $z = \infty$, the logarithms exist. Let us denote them as

$$\ln x(z) = y(z), \quad \ln m(z) = n(z) \ .$$

Here $n(z)$, obtained from the given $m(z)$, is a symmetric (infinite) power series

$$n(z) = \cdots + n_1 z + n_0 + n_1 z^{-1} + \cdots \ .$$

It can be easily decomposed,

$$n(z) = y(z) + y(z^{-1})$$

with power series

$$y(z) = y_0 + y_1 z^{-1} + \cdots = \frac{n_0}{2} + n_1 z^{-1} + \cdots \ ,$$

analytic for $1 - \varepsilon < |z|$. Once $y(z)$ is computed, the spectral factor $x(z)$ is recovered as

$$x(z) = \mathrm{e}^{y(z)} = x_0 + x_1 z^{-1} + \cdots . \tag{6}$$

Since $y(z)$ is analytic in $1 - \varepsilon < |z|$, so is $x(z)$ and hence it can be expanded according to (3). Moreover, as a result of exponential function, $x(z)$ is nonzero in $1 - \varepsilon < |z|$. In other words, it has all its zeros inside the unit disc and is therefore Schur stable. Note also that $x(z)$ has to be a (finite) polynomial of degree d (due to the uniqueness of the solution to the problem which is known to be a polynomial) though $y(z)$ is an infinite power series.

Numerical implementation follows the ideas considered above. A polynomial $p(z)$ is represented by its coefficients p_i, $i = 0 \ldots r$ or, equivalently, by function values P_k in the Fourier interpolating points g^k, $k = -R \ldots 0 \ldots R$, where $R \geq d$, $\quad g = \mathrm{e}^{\mathrm{j}\frac{2\pi}{2R+1}}$. Accordingly, a power series can be approximated by a finite set of its coefficients or by its values in a finite number of interpolation points on the unit circle. Some operations of the procedure, namely the decomposition of $n(z)$ into $y(z)$ and $y(z^-1)$, are performed in the time domain (operations on coefficients), while the others (evaluation of logarithmic and exponential functions) are executed in the frequency domain (operations with values over $|z| = 1$). Mutual conversion between the two domains is mediated by the shifted discrete Fourier transform operator defined as

$$X_k = \sum_{i=-R}^{R} x_i g^{-ki}, \quad x_i = \frac{1}{2R+1} \sum_{k=-R}^{R} X_k g^{ki} \ ,$$

which approximates the Z-transform by dealing with $-R \leq i \leq +R$ instead of infinite $-\infty < i < +\infty$, and with $z = g^k, \ -R \leq k \leq +R$ instead of continuum $z = \mathrm{e}^{\mathrm{j}\phi}, \ -\pi \leq \phi \leq +\pi$.

The accuracy of results depends on the number of interpolation points $2R+1$ involved in the computation. This number can be considered as a simple tuning knob of the computational process.

The resulting numerical routine, described in detail in [Ježek et al., 2000], looks then as follows:

Input: Scalar symmetric polynomial
$m(z) = m_d z^d + \cdots + m_1 z + m_0 + m_1 z^{-1} + \cdots + m_d z^{-d}$, positive for $|z| = 1$.

Output: Polynomial $x(z) = x_0 + x_1 z^{-1} + \cdots + x_d z^{-d}$, the spectral factor of $m(z)$.

Step 1 - *Choice of the number of interpolation points.*
Decide about the number R. R approximately 10 to 50 times larger than d is recommended up to our practical experience.

Step 2 - *Direct FFT (I):*
Using the FFT algorithm, perform direct DFT, defined by (1), on the vector

$$\boldsymbol{m} = \underbrace{[m_0, m_1, \ldots, m_n, 0, 0, \ldots, 0, m_n, \ldots, m_1]}_{2R+1}$$

In this way, the set $\boldsymbol{M} = [M_0, M_1, \ldots, M_{2R}]$ of the values of $m(z)$ at the Fourier points is obtained. Owing to the symmetry of $m(z)$ and assuming $m(z)$ with real coefficients only, M is symmetric and real as well.

Step 3 - *Logarithmization:*
Compute the logarithms $N_i = \ln(M_i)$ of all particular M_i's and form the vector $\boldsymbol{N} = [N_0, N_1, \ldots, N_{2R}]$ of them. N_i's thus obtained are the values of the function $n(z) = \ln(m(z))$ at related Fourier points on the unit complex circle.

Step 4 - *Inverse FFT (I):*
To get the vector
$\boldsymbol{n} = [n_0, n_1, \ldots, n_R, n_R, \ldots, n_1]$, containing the coefficients of the two-sided symmetric polynomial $n(z) = n_R z^{-R} + \cdots + n_1 z^{-1} + n_0 + n_1 z + \cdots + n_R z^R$ approximating the power series $\ln(m(z))$ for the given R, perform inverse DFT, defined by (2), on the vector $\boldsymbol{N}$ using the FFT algorithm.

Step 5 - *Decomposition:*

Take the "causal part" $\boldsymbol{y}$ of $\boldsymbol{n}$:
$\boldsymbol{y} = [n_0/2, n_1, \ldots, n_R]$.

Step 6 - *Direct FFT (II):*

Evaluate $y(z) = n_0/2 + n_1 z^{-1} + \ldots + n_R z^{-R}$ at the Fourier points by applying direct FFT on the set

$$\underbrace{[\frac{n_0}{2}, n_1, \ldots, n_R, 0, 0, \ldots, 0]}_{2R+1}$$

and get $\boldsymbol{Y} = [Y_0, \ldots, Y_{2R}]$.

Step 7 - *Exponential function:*

To get the spectral factor, the exponential function $x(z) = e^{y(z)}$ remains to be evaluated. First we compute the values of $x(z)$ at the Fourier points: $\boldsymbol{X} = [e^{Y_0}, \ldots, e^{Y_{2R}}]$.

Step 8 - *Inverse FFT (II):*

Finally, the coefficients $\boldsymbol{x} = [x_0, \ldots, x_{2R}]$ of $x(z)$ are recovered by inverse FFT performed on the vector $\boldsymbol{X}$. The resulting approximation to the spectral factor $x(z)$ then equals $x(z) = x_0 + x_1 z^{-1} + \cdots + x_d z^{-d}$. $\diamond$

Numerical experiments prove the efficiency of the DFT approach. In table 4.1 some results are presented for a particular large-degree polynomial which arises in a certain signal processing application, see [Sternad et al., 2000] for further details. The data was kindly provided by Mikael Sternad and colleagues from the University of Uppsala, Sweden.

The Method ITER(N) is the Newton-Raphson iterative method programmed in the function `spf` of the Polynomial Toolbox for Matlab, with given number of iterations N. The Method FFT(R) is the method described here. The R parameter determines the number 2^R of Fourier interpolation points used in particular runs. The computational error is defined here as the largest coefficient of the expression $xx^* - m$, evaluated in the Matlab workspace, divided by the largest coefficient of m (all in absolute value).

	Time [s]	Flops	Accuracy
ITER(7)	18 sec	$7.8 \cdot 10^8$	$5 \cdot 10^{-5}$
FFT(10)	0.01 sec	$9.7 \cdot 10^4$	$5 \cdot 10^{-5}$
ITER(8)	20 sec	$1.4 \cdot 10^9$	$4 \cdot 10^{-6}$
FFT(11)	0.02 sec	$3.6 \cdot 10^5$	$3 \cdot 10^{-6}$
ITER(9)	22 sec	$1.6 \cdot 10^9$	$3 \cdot 10^{-8}$
FFT(12)	0.05 sec	$1.6 \cdot 10^6$	$2 \cdot 10^{-8}$
ITER(10)	26 sec	$1.8 \cdot 10^9$	10^{-11}
FFT(13)	0.11 sec	$5.5 \cdot 10^6$	$2 \cdot 10^{-11}$

Table 4.1: Accuracy and efficiency of compared algorithms.

5 Polynomial Matrix Polytopes

Stability of polynomial matrix polytopes is directly related to robust stability of linear systems described by matrix fraction descriptions and affected by polytopic uncertainty. Polytopic uncertainty is probably the most general representation of the parametric uncertainty affecting a linear system. However, this generality has recently led to the somehow disappointing conclusion that most of the stability analysis problems involving polytopic uncertainty cannot be solved efficiently.

In [Henrion et al., 2000], it is shown that sufficient robust stability conditions can be derived via parameter dependent Lyapunov functions. These conditions are expressed as convex LMI problems, whereas the original necessary and sufficient stability conditions unavoidably lead to a non-convex problem. In this section, we will illustrate these results with the help of a simple numerical example.

Consider the continuous-time interval polynomial matrix

$$\left[\begin{array}{cc} [7.7-2.3s+4.3s^2, 3.7+2.7s+4.3s^2] & [-3.1-6s-2.2s^2, -4.1-7s-2.2s^2] \\ 3.6+6.4s+4.3s^2 & [3.2+11s+8.2s^2, 16+12s+8.2s^2] \end{array}\right].$$

Associated to the above interval polynomial matrix is a polynomial matrix polytope of degree $d=2$ with $N=2^3=8$ vertices denoted by $A_1(s),\ldots,A_8(s)$.

As described in [Henrion et al., 2000], the polytope of polynomial matrices with vertices $A_i(s)$ is robustly stable if there exist matrices $P_1,\ldots,P_8$ and F,G solution to the LMI feasibility problem

$$\begin{array}{ll} \left[\begin{array}{c} \mathcal{Q} \\ \mathcal{P} \\ \mathcal{A}_i \end{array}\right]^T \left[\begin{array}{ccc} 0 & P_i & F \\ P_i & 0 & G \\ F^T & G^T & 0 \end{array}\right] \left[\begin{array}{c} \mathcal{Q} \\ \mathcal{P} \\ \mathcal{A}_i \end{array}\right] \prec 0 & \quad i=1,\ldots,N \\ P_i = P_i^T \succ 0 & \end{array} \tag{7}$$

where

$$\mathcal{P} = \left[\begin{array}{cccccc} 0&0&1&0&0&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{array}\right] \qquad \mathcal{Q} = \left[\begin{array}{cccccc} 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&0&0 \end{array}\right]$$

are projection matrices, and

$$\mathcal{A}_i = \left[\begin{array}{cccc} A_i^0 & A_i^1 & \cdots & A_i^d \end{array}\right]$$

are constant matrices built from coefficients of polynomial matrix vertices

$$A_i(s) = A_i^0 + A_i^1 s + \cdot + A_i^d s^d.$$

Using the LMI Toolbox for Matlab, the above problem was found feasible for

$$P_1 = \left[\begin{array}{cccc} 3.7354 & 0.2711 & 0.1767 & 0.0364 \\ 0.2711 & 2.2550 & 0.4162 & 1.0318 \\ 0.1767 & 0.4162 & 2.7773 & 1.2283 \\ 0.0364 & 1.0318 & 1.2283 & 2.8336 \end{array}\right] P_2 = \left[\begin{array}{cccc} 2.7504 & -0.0435 & 1.0393 & 0.3271 \\ -0.0435 & 2.5026 & -0.0255 & 1.0368 \\ 1.0393 & -0.0255 & 2.9039 & 0.7286 \\ 0.3271 & 1.0368 & 0.7286 & 2.8959 \end{array}\right] \cdots$$

and

$$F = \begin{bmatrix} -0.1267 & -0.1034 \\ 0.1395 & -0.0547 \\ 213.0715 & -17906.3902 \\ 4445.8848 & 11925.2280 \end{bmatrix} \quad G = \begin{bmatrix} -213.4209 & 17906.1930 \\ -4445.7996 & -11925.3861 \\ -0.0465 & -0.0906 \\ 0.1415 & -0.0472 \end{bmatrix}.$$

The root locus of the interval polynomial matrix, obtained by a brute-force three-dimensional sweeping, is represented in figure 5.1. One can check that the interval polynomial is indeed robustly stable.

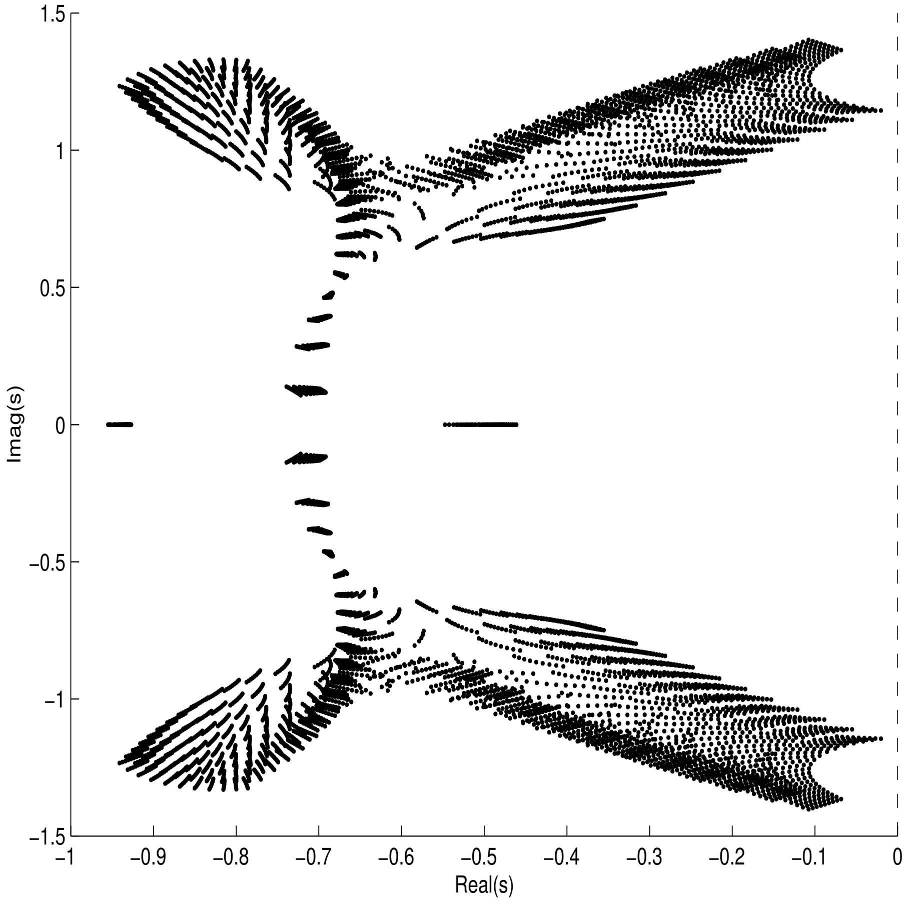

Figure 5.1: Root Locus of the Interval Polynomial Matrix

Numerical experiments were then carried out to evaluate the amount of conservatism of sufficient robust stability conditions (7) for scalar polynomials, i.e. $n = 1$. In this case, we can easily and exactly check robust stability of polynomial polytopes with the help of the Edge Theorem. We randomly generated 100 polytopes of polynomials for different values of degree d and number of vertices N. Each vertex of each polytope was chosen as a random stable monic continuous-time polynomial generated with macro `prand` of the Polynomial Toolbox. Of course, stability of each vertex does not imply robust stability of the whole polytope. In table 5.1 we reported the number of polytopes that were found robustly stable

- with quadratic stability condition obtained by setting $P_i = P$ for $i = 1, \ldots, N$ in LMI (7) (columns labeled QS)

- with our new stability condition (7) (columns labeled NS)
- with the Edge Theorem (columns labeled RS).

From the figures in table 5.1, we can see that our new stability condition scarcely appeared conservative for low values of d. This is in sharp contrast with the quadratic stability condition.

d	N	QS	NS	RS	d	N	QS	NS	RS	d	N	QS	NS	RS
2	2	86	100	100	4	2	51	92	92	6	2	38	73	79
2	3	68	100	100	4	3	15	73	73	6	3	7	54	58
2	4	52	100	100	4	4	5	73	73	6	4	3	38	46
2	5	43	100	100	4	5	3	61	61	6	5	0	21	36
2	6	29	100	100	4	6	3	52	53	6	6	0	4	20
2	7	27	100	100	4	7	0	49	52	6	7	0	3	17
3	2	55	98	98	5	2	34	83	83	7	2	20	70	77
3	3	31	96	96	5	3	11	72	72	7	3	4	34	49
3	4	20	95	95	5	4	3	56	57	7	4	0	22	34
3	5	7	85	85	5	5	1	42	44	7	5	0	9	22
3	6	6	81	81	5	6	0	29	31	7	6	0	0	10
3	7	4	78	78	5	7	0	22	26	7	7	0	0	3

Table 5.1 - Stable polytopes with different stability criteria.

References

[Boyd et al., 1994] S. Boyd, L. El Ghaoui, E. Feron and V. Balakrishnan "Linear Matrix Inequalities in System and Control Theory", *SIAM Studies in Applied Mathematics*, Philadelphia, Pennsylvania, 1994.

[Henrion and Šebek, 1999] D. Henrion and M. Šebek "Reliable Numerical Methods for Polynomial Matrix Triangularization", *IEEE Transactions on Automatic Control*, Vol. 44, No. 3, pp. 497–508, 1999.

[Henrion et al., 2000] D. Henrion, D. Arzelier, D. Peaucelle and M. Šebek "An LMI Condition for Robust Stability of Polynomial Matrix Polytopes", *Proceedings of the IFAC Symposium on Robust Control Design*, Prague, Czechia, 2000. Also to appear in *Automatica*, Vol. 37, No. 3, 2001.

[Hromčík and Šebek, 1999] M. Hromčík and M. Šebek "New Algorithm for Polynomial Matrix Determinant based on FFT", *Proceedings of the European Control Conference*, Karlsruhe, Germany, 1999.

[Ježek et al., 2000] J. Ježek, M. Hromčík and M. Šebek "Spectral Factorization by Means of Discrete Fourier Transform", *Proceedings of the IEEE Mediterranean Conference on Control and Automation*, University of Patras, Greece, 2000.

[Kučera, 1979] V. Kučera "Discrete Linear Control: The Polynomial Approach", *John Wiley and Sons*, Chichester, 1979.

[Labhalla et al., 1996] S. Labhalla, H. Lombardi and R. Marlin "Algorithmes de Calcul de la Réduction de Hermite d'une Matrice à Coefficients Polynomiaux", *Theoretical Computer Science*, Vol. 161, pp. 69–92, 1996.

[The Mathworks, 1998] The MathWorks, Inc. "LMI Control Toolbox for Matlab", Release 1.0.5, 1998.

[PolyX, 2000] PolyX, Inc., "Polynomial Toolbox for MATLAB", see the web page `www.polyx.cz`, 2000.

[Sternad et al., 2000] M. Sternad, M. Johansson and J. Rutstrom "Inversion of Loudspeaker Dynamics by Polynomial LQ Feedforward Control", *Proceedings of the IFAC Symposium on Robust Control Design*, Prague, Czech Republic, 2000.