POLYNOMIAL DESIGN METHODS AND A SIGNAL PROCESSING APPLICATION [*]

Martin Hromčík, Michael Šebek

Centre for Applied Cybernetics,
Czech Technical University, Prague, Czech Republic

Abstract

The use of the MATLAB environment and the Polynomial Toolbox for Matlab capabilities are demonstrated in a specific signal processing application. A new algorithm is described for the spectral factorization of a two-sided symmetric polynomial. The method is based on the discrete Fourier transform theory (DFT) and its relationship to the Z-transform. Involving DFT computational techniques, namely the famous fast Fourier transform routine (FFT), brings high computational efficiency and reliability. The power of the proposed procedure is employed in a particular practical application. Namely the problem of computing an H_2-optimal inverse dynamic filter to an audio equipment is considered as it was proposed by M. Sternad and colleagues in [17] to improve behavior of moderate quality loudspeakers. Involved spectral factorization is resolved by our new method and its performance is compared with existing algorithms.

1 Introduction

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

A matrix equivalent to the scalar case is the computation of the spectral factor $X(z) = X_0 + X_1 z^{-1} + \cdots + X_d z^{-d}$ to a discrete-time para-Hermitian two-sided polynomial matrix

$$M(z) = \sum_{i=0}^{d} (M_i z^i + \overline{M}_i^T z^{-i}),$$

nonsingular for $|z| = 1$. Here $X(z)$ is required to be Schur-stable again and to fulfill the equation $M(z) = X(z)\overline{X^T(\bar{z}^{-1})}$.

[*]This work has been supported by the Ministry of Education of the Czech Republic under contract No. LN00B096.

If we restrict ourselves to real coefficients only, the formulas for the adjoint operator reduce to $m^{\sim}(z) = m(z^{-1}) = \sum_{i=-d}^{d} m_i z^{-i}$ and $M^{\sim}(z) = M^T(z^{-1}) = \sum_{i=-d}^{d} M_i^T z^{-i}$ respectively whereas the symmetry condition reads $m_{-i} = m_i$ or $M_{-i} = M_i^T$ respectively.

The spectral factorization is one of the basic operations in the polynomial approach to the synthesis of linear control systems [4, 5] and filters [15, 16, 17]. It is a part of the procedure of quadratic functional minimization under a condition of causality. Particular forms of the functional to be minimized lead to some well known control design problems, such as Wiener filtering, linear-quadratic controller (LQ), quadratic optimal observer (Kalman filter), linear-quadratic controller with Gaussian noise (LQG) and some others. All these tasks, being solved via polynomial methods, involve spectral factorization as the crucial computational step [4, 5].

2 Existing methods

In any case, the spectral factor for $d \geq 5$ cannot be achieved by a finite number of algebraic operations. Therefore all numerical algorithms for its computation are iterative and give just an approximation to the genuine factor. Some existing approaches to this problem are mentioned in this section.

The most natural way is based on the computation of roots of a polynomial. Obviously, $m(z)$ being a symmetric two-sided polynomial yields $m^\star(z) = z^{-d}m(z)$ to be a one-sided polynomial in z^{-1} with roots symmetric with respect to the unit circle in the complex plane. The roots of $m^\star(z)$ (all are nonzero) are the zeros of $m(z)$.

This idea can be directly used to evaluate the spectral factor. Having determined the roots $r_1, r_2, \ldots, r_d$ of $m^\star(z)$ via any standard procedure for polynomial roots [3] and considering that $m(z) \neq 0$ for all $|z| = 1$ by assumption, one can divide the roots into two groups $R^- = \{r_i : m^\star(r_i) = 0, |r_i| < 1\}, R^+ = \{r_i : m^\star(r_i) = 0, |r_i| > 1\}$. Clearly, R^- is the set of roots of the spectral factor $x(z)$ and having this set at hand the factor itself can be easily constructed.

Taking the symmetry of roots into account, a more sophisticated procedure can be suggested in the real case. By introducing a new variable $y = z + z^{-1}$, the two sided symmetric polynomial

$$m(z) = m_d z^{-d} + \cdots + m_0 + \cdots + m_d z^d$$

can be rewritten as

$$m(z) = p(y) = p_0 + p_1 y + \cdots + p_d y^d$$

with p being a one-sided polynomial only. In this way the degree of input is reduced by one half. The coefficients of $p(y)$ and $m(z)$ are linearly dependent.

At any rate, performance of this procedure heavily hinges on the accuracy of the computed polynomial roots. If these roots are distinct and separated enough, standard numerical routines [3] can determine them with good precision. However, it is well known that the relative accuracy of a computed root decreases as its multiplicity grows [3], and so does the accuracy of the spectral factor thus obtained.

In addition, if the degree of the involved polynomial is high, say over one hundred, the very computation of the spectral factor coefficients is problematic due to rounding errors. It means that even if the desired *roots* of the spectral factor are evaluated with good accuracy, its particular *coefficients*, which are typically required in applications, are not accurate.

Needless to say, this approach is not directly applicable in the matrix case.

For these reasons, attention was paid to the development of other routines that avoid the direct roots evaluation. One such procedure, relying on successive Newton-Raphson iterations and solutions of symmetric polynomial equations, was published in [11] and is implemented in the Polynomial Toolbox 2.0 for Matlab [6]. Other methods are based on the formulation of the problem in terms of the state-space theory and its solution via algebraic Riccati equations, or employ interpolation [9]. Recently some new approaches to spectral factorization have appeared based on the analysis of quadratic forms [10]. All these techniques can also be extended to the matrix case.

In the next sections we will described a new approach to the problem. It is based on the DFT theory and provides both a fruitful view on the relation between DFT and the Z-transform theory, and a powerful computational tool in the form of the fast Fourier transform algorithm.

3 Discrete Fourier Transform

For a vector of complex numbers, DFT is defined as follows:

Definition 1 (see [2, 1]) - *direct DFT*:
If $\boldsymbol{p} = [p_0, p_1, \ldots, p_N]$ is a vector of complex numbers, then its *direct DFT* is given by the vector $\boldsymbol{y} = [y_0, y_1, \ldots, y_N]$, where

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

The vector $\boldsymbol{y}$ is called the image of vector $\boldsymbol{p}$. ◇

Definition 3.2 (see [2, 1]) - *inverse DFT*:
If $\boldsymbol{y} = [y_0, y_1, \ldots, y_N]$ is a vector of complex numbers, then its inverse *DFT* is given by a vector $\boldsymbol{p} = [p_0, p_1, \ldots, p_N]$, where

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

◇

If $\boldsymbol{y}$ is an image of $\boldsymbol{p}$, then the formula (2) returns the original vector $\boldsymbol{p}$ [2, 1]. Hence the relation (2) is inverse to relation (1).

DFT is of great interest in various engineering fields. For its relationship to Fourier series of sampled signals, DFT is frequently used in signal processing [2]. One of the experimental identification methods employs DFT as well [7]. The close relationship of DFT to interpolation is also well known and was used recently to solve some tasks of the polynomial control theory [12, 13] and to treat robustness analysis problems of certain kind [14].

For numerical computation of DFT, the efficient recursive FFT algorithm was developed by Cooley and Tukey in 1965 [2], [1]. If the length of the input is a power of two, a faster version of FFT (sometimes called *radix-2 FFT*) can be employed [2, 1]. In general, the FFT routine features a highly beneficial computational complexity and involves $\mathcal{O}(N \log(N))$ multiplications and additions for a vector of length N.

For the importance of DFT mentioned above, the FFT algorithms are naturally available as built-in functions of many computing packages (MATLAB$^{\text{TM}}$, MATHEMATICA$^{\text{TM}}$ etc.). This is another good reason for employing the procedure proposed in this paper.

4 Spectral Factorization and DFT

4.1 Theory

Given a 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 $|z| = 1$, we look for a Schur stable polynomial

$$x(z) = x_0 + x_1 z^{-1} + \cdots + x_d z^{-d}$$

to satisfy

$$x(z)x(z^{-1}) = m(z) \ .$$

In order to solve the 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

$$x(z)=\mathrm{e}^{y(z)}=x_0+x_1z^{-1}+\cdots. \tag{3}$$

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.

4.2 Numerical Algorithm

Numerical implementation follows the ideas considered above. A polynomial $p(z)$ is represented by its coefficients $p_i,\ i=0\dots r$ or, equivalently, by function values P_k in the Fourier interpolating points $g^k,\ k=-R\dots0\dots R$, 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

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

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.

Resulting numerical routine looks then as follows:

Algorithm 1: Scalar discrete-time spectral factorization.

Input: Scalar symmetric polynomial
$m(z)=m_dz^d+\cdots+m_1z+m_0+m_1z^{-1}+\cdots+m_dz^{-d}$, positive for $|z|=1$.

Output: Polynomial $x(z)=x_0+x_1z^{-1}+\cdots+x_dz^{-d}$, 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

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

In this way, the set $\boldsymbol{M}=[M_0,M_1,\dots,M_{2R}]$ 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 $\boldsymbol{N}=[N_0,N_1,\dots,N_{2R}]$ 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
$\boldsymbol{n}=[n_0,n_1,\dots,n_R,n_R,\dots,n_1]$, containing the coefficients of the two-sided symmetric polynomial $n(z)=n_Rz^{-R}+\cdots+n_1z^{-1}+n_0+n_1z+\cdots+n_Rz^R$ 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 $x(z) = x_0 + x_1 z^{-1} + \cdots + x_d z^{-d}$. ◇

Note that one obtains $2R+1$ coefficients of $x(z) = x_0 + x_1 z^{-1} + \cdots x_{2R} z^{-2R}$ in the Step 8. However, $x(z)$ being the spectral factor of $m(z)$ is known to be of degree d only and only the first $d+1$ coefficients of $x(z)$ should be significant as a result while the remaining ones should be negligible. As the number R increases, these values theoretically converge to zero indeed since the formulas of DFT become better approximations to the Z-transform definitions.

5 Computational Complexity

Thanks to the fact that the fast Fourier transform algorithm is extensively used during the computation, the overall routine features a expedient computational complexity.

Provided that the above modifications of the computational procedure are considered, namely if the resulting number of interpolation points is taken as a power of two, the fast radix-2 FFT can be employed. In this case, $(R \log_2 R)/2$ multiplications and $R \log_2 R$ additions are needed to evaluate either direct or inverse DFT of a vector of length R [1]. Let us suppose in addition that computing the logarithm or exponential of a scalar constant takes at most k multiplications and l additions. Then the particular steps of the modified Algorithm 1 involve $(R \log_2 R)/2$ multiplications and $R \log_2 R$ additions (Steps 2, 4, 6, 8), and kR multiplications and lR additions (Steps 3, 7) respectively. Hence the overall procedure consumes

$$4\frac{R \log R}{2} + 2kR = 2R \log R + 2lR$$

complex multiplications, and

$$4R \log R + 2lR$$

complex additions. By inspecting the above formulas one can see that asymptotically the proposed method features $\mathcal{O}(R \log R)$ complex multiplications and additions.

6 Implementation and Practical Experience

The algorithm has been implemented in MATLAB, using the standard routine for the fast Fourier transform (`fft` command). This program has also been evaluated by the PolyX company, the producer of the Polynomial Toolbox for Matlab [6], and a decision was made recently to incorporate the code in the spectral factorization solver of the Toolbox in the next version 3.

The practical experience is very good. Though the complete routine seemingly issues a high number of manipulations, it is pretty fast thanks to the computational efficiency of the FFT algorithm. It runs many times faster than the Newton-Raphson iterations procedure as it is programmed in the Polynomial Toolbox 2.0 for Matlab [6], returning results of comparable accuracy. The execution times are similar with the method based on direct evaluation of roots, however, in the case of multiple roots, and namely for high degrees, the accuracy of the newly proposed method is much better provided the number of interpolation points is adequately chosen.

These observations are based on practical experiments with real life data. To be more specific, we present one practical signal processing application in the next section involving the spectral factor extraction and provide its effective solution via our new routine.

7 Upgrading Loudspeakers Dynamics

An original approach has been published by Sternad et al. in [17] how to improve performance of an audio equipment at low additional costs. The authors use the LQG optimal feedforward compensator technique to receive an inverse dynamic filter for a moderate quality loudspeaker. By attaching a signal processor implementing this filter prior to the loudspeaker, the dynamical imperfections of the original device are eliminated and the overall equipment behaves as an aparatus of a much higher class. To learn more about this research and to get some working examples, visit [18].

Unlike their predecessors, the authors try to modify the sound over the whole range of frequencies. Such a complex compensation fully employs the increasing performance of signal hardware dedicated to CD-quality audio signals, and at the same time calls for fast and reliable spectral factorization solvers [17]. We believe our new algorithm will significantly contribute to this goal.

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

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

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

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

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

such that the criterion

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

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

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

where β results from the spectral factorization

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

and Q_1 is the solution of a subsequent Diophantine equation

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

see [17].

As for the spectral factor computation, the authors employ the Newton-Raphson iterative scheme [11] in the cited work [17]. According to their results and our experience, this method has been probably the best available procedure for scalar polynomial spectral factorization so far [17, 6]. This method works quite well also for high degrees of involved polynomials in contrast to the straightforward way of computing and distributing the roots of $BB^* + \rho AA^*$.

Let us perform a benchmark experiment to compare the existing approach and our newly proposed algorithm for particular numerical data kindly provided by Mikael Sternad and colleagues from the University of Uppsala. In particular, $B(z) = B_0 + B_1 z^-1 + \cdots + B_250 z^{-250}$ is an unstable polynomial of degree 250, $A(z)$ is stable of degree 90 and $k = 160$. Taking $\rho = 0$, the spectral factorization of $m(z) = B(z)B^*(z) = m_{250}z^{-250} + \ldots + m_0 + \ldots + m_{250}z^{250}$ is to be performed.

All presented experiments were realized on a PC computer with Pentium II/400MHz processor and 128 MB RAM, under MS Windows 98 in MATLAB version 5.3. Implementation of particular routines is specified below:

- Method ITER(N): The Newton-Raphson iterative method is programmed in the function `spf` of the Polynomial Toolbox for Matlab [11]. This macro is used in our experiment. Several cases are considered depending on the number of iterations N.
- Method FFT(R): The method which is the subject of this paper has given rise to a function programmed in the MATLAB programming language. The R parameter determines the number 2^R of Fourier interpolation points used in particular runs.

Results of this experiment for various values of the parameters N and R are summarized and related in the following table. Namely, the computational time, total number of floating point operations and accuracy of results are of interest. To obtain the former two characteristics, the MATLAB abilities were employed (the built-in functions `tic/toc` and `flops` respectively). The computational error is defined here as the largest coefficient of the expression $\beta\beta^* - m$, 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 1: Accuracy and efficiency of compared algorithms.

These tests prove the power of the new algorithm in such tough examples. By checking the above table, one can see that it typically consumes more than 100 times less computational time than its competitor and involves 1000 times fewer elementary operations to achieve comparable accuracy of results.

Just for completeness, the results for direct-roots-evaluation approach are also presented. The symmetry of $m(z)$ was taken into account to reduce its degree by the transform $z \to z + z^{-1}$ as described in section 2. The roots were evaluated as the eigenvalues of related companion matrix by the Matlab command `roots`, see [19].

	Time [s]	Flops	Accuracy
ROOTS	2 sec	$1.6 \cdot 10^8$	**0.9**

Table 1b: Accuracy and efficiency of the direct method.

As it was pointed out before, the direct method cannot be recommended for high degrees since its precision becomes poor. In fact, the high accuracy value 0.9 in the Table 1b means that the error of computation is of about the same magnitude as the input which is a hardly acceptable result (compare with Table 1a). In addition, the method is rather slow in comparison with the FFT based routine.

8 Conclusion

A new method for the discrete-time spectral factorization problem in the scalar case has been proposed. The new method relies on numerically stable and efficient FFT algorithm. Besides its good numerical properties, the derivation of the routine also provides an interesting look into the related mathematics, combining the results of the theory of functions of complex variable, the theory of sampled signals, and the discrete Fourier transform techniques. Its numerical properties are discussed with respect to other existing algorithms and its power is utilized in a practical signal processing application.

References

[1] Bini D., Pan V., *Polynomial and Matrix Computations, Volume 1: Fundamental algorithms*, Birkhäuser, Boston (1994).

[2] Čížek V., *Discrete Fourier Transforms and Their Applications*, Adam Hilger Ltd, Bristol and Boston (1986).

[3] Higham N. J., *Accuracy and Stability of Numerical Algorithms*, S.I.A.M., Philadelphia (1996).

[4] Kailath T., *Linear Systems*, Prentice Hall, New Jersey (1980).

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

[6] Kwakernaak H., Šebek M., *PolyX Home Page*, `http://www.polyx.cz/`, `http://www.polyx.com/`.

[7] Ljung L., *System Identification: Theory for the User*, Prentice-Hall Information and Systems Sciences Series. Englewood Cliffs, Prentice-Hall (1987).

[8] Kwakernaak H., Šebek M, *Polynomial J-Spectral Factorization*, IEEE Trans. Automatic Control, Vol. 39, No.2, pp. 315-328 (1994).

[9] Green M., Glover K, Limebeer D. and Doyle J., *A J-spectral Factorization Approach to H_∞ control*, SIAM J. on Contr. Opt., vol. 28, pp. 1350-1371 (1990).

[10] Kaneko O. and Fujii T., *Discrete Time Behavioral Dissipativeness and Spectral Factorization via Quadratic Difference Forms*, Proceedings of the 5th European Control Conference ECC'99, Karlsruhe, Germany, October 31 - September 3 (Session BA9), 1999.

[11] Ježek J. and Kučera V., *Efficient Algorithm for Matrix Spectral Factorization,*Automatica, vol. 29, pp. 663-669, 1985.

[12] Hromčík M., Šebek M., *New Algorithm for Polynomial Matrix Determinant Based on FFT*, Proceedings of the 5th European Control Conference ECC'99, Karlsruhe, Germany, October 31 - September 3 (Session DA1), 1999.

[13] Hromčík M., Šebek M., *Numerical and Symbolic Computation of Polynomial Matrix Determinant*, Proceedings of the 38th Conference on Decision and Control CDC'99, Phoenix AZ, USA, December 7-10, 1999.

[14] Hromčík M., Šebek M, *Fast Fourier Transform and Robustness Analysis with Respect to Parametric Uncertainties*, Proceedings of the 3rd IFAC Symposium on Robust Control Design ROCOND 2000, Prague, CZ, June 21-23, 2000.

[15] M. Sternad and A. Ahleén, *Robust Filtering and Feedforward Control Based on Probabilistic Descriptions of Model Errors*, Automatica, 29, pp. 661-679.

[16] K. Ohrn, A. Ahleén and M. Sternad, *A Probabilistic Approach to Multivariable Robust Filtering and Open-loop Control*, IEEE Transactions on Automatic Control, 40, pp. 405-417.

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

[18] www.signal.uu.se/Staff/ms/ms.html

[19] Using MATLAB 5.3, The Matrhworks, 1999.

[11] H. Kwakernaak and M. Šebek: *Polynomial Toolbox Home Page*, `http://www.polyx.cz`, `http://www,polyx.com`