Semisymbolic Analysis of Linear Systems using MATLAB

Dalibor Biolek and Viera Biolková
Faculty of Electrical Engineering and Informatics
Brno University of Technology
Purkyňova 118, 612 00 Brno, Czech Republic
biolek@cs.vabo.cz

Abstract

Some new M-files are described which help the MATLAB users to analyze linear systems using the semisymbolic approach.

1 Introduction

The symbolic/semisymbolic analysis of linear systems specifies such form of result, where a computed system function (e.g. input-output transfer, imittances, two-port parameters,..) is derived using its coefficients as symbols/numbers. Semisymbolic result in the *s*-domain is a useful gateway for following numerical analyses: zero/pole and stability analysis, frequency analysis (after substitution $s = j\omega$), semisymbolic and numerical time-domain analysis (after partial fraction expansion of system function), sensitivity analysis, etc.

Several MATLAB functions support semisymbolic and numerical analyses: *EIG* and *QZ* for solving eigenvalue problem, *POLY* for computing coefficients of system functions from system matrices or zeros/poles, *ROOTS* for zeros/poles computation from coefficients, *TF2ZP* for recounting zeros and poles from coefficients of system function, *ZP2TF* for recounting coefficients from zeros and poles, *RESIDUE* for partial fraction expansion, *FREQS* for computation of frequency responses, etc. [1].

It will be shown that not all of these functions are optimal for our purposes. Part of them can be improved, and we also compiled some of new ones.

2 Utilization Of Current Functions

Fig. 1 illustrates possible utilization of some aforementioned MATLAB functions for the analysis of linear systems based on the semisymbolic approach. Let us explain the conception on the example of electrical systems, but it can be used for solution of arbitrary linear systems. Let us consider input data in the form of a set of *s*-domain linear equations with a square system matrix

$$\mathbf{Y}(s) = \mathbf{G} + s\mathbf{C}. \qquad (1)$$

For electrical systems, matrix (1) can represent the modified nodal analysis (MNA) matrix [2]. Suppose that a computed system function is the ratio of two algebraic minors of the system matrix (1). Fig. 1 considers processing of a matrix $\widetilde{\mathbf{G}} + s\widetilde{\mathbf{C}}$ which corresponds to one of this minor.

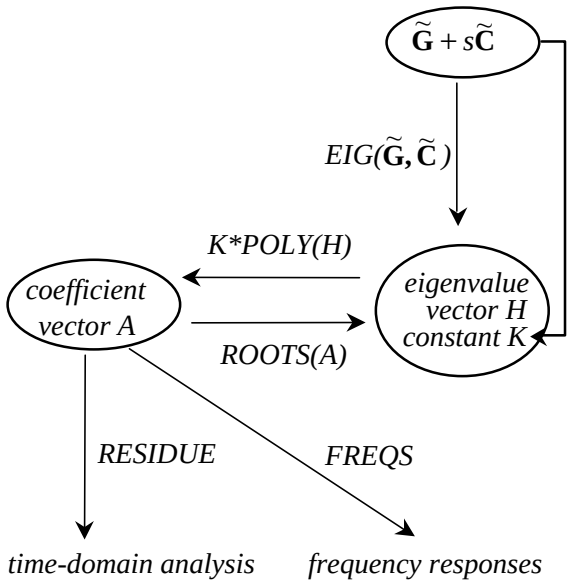

Figure 1. Utilization of standard MATLAB functions.

The basic step of the semisymbolic analysis is to find numerical coefficients a_i or roots r_i and constant *K* of the minor determinant

$$\det(\widetilde{\mathbf{G}} + s\widetilde{\mathbf{C}}) = \sum_{i=0}^{n} a_i s^i = K\prod_{i=1}^{n}(s - r_i). \qquad (2)$$

Here *n* is a number of roots which can be smaller that the minor order *N*. To decrease numerical inaccuracies, it is preferable to compute coefficients from the roots than vice versa [3]. The standard MATLAB function

$$H = EIG\left(\tilde{\mathbf{G}}, \tilde{\mathbf{C}}\right) \quad (3)$$

can be applied for a solution of this generalized eigenvalue problem. In MATLAB notation, number of elements of a generated vector H is N, even if the actual number of eigenvalues is $n \leq N$. The N-n elements in vector H are then generated in a "*NaN*" format (not a number). Testing these elements in a cycle, we can exclude them from the vector of eigenvalues and to find the order n.

The constant K can be obtained using a simple auxiliary procedure: we substitute an arbitrary complex number s to the original minor matrix, then we compute the minor and compare it with right-side product in (2) after the same substitution. This constant has to be principally real.

The coefficient vector A can be generated from roots by a standard MATLAB function *POLY*:

$$A = K * POLY(H) \quad (4)$$

However, the result of this operation can be entirely false if the roots are computed with a small error.

In some cases, the input data of the analysis are represented by the coefficients of system function (or in other words, by the coefficient of a corresponding input/output differential equation). Then recalculation to zeros and poles is often required. This operation can be a source of serious numerical problems, especially in case of multiple roots and other ill-conditioned polynomials [4]. MATLAB offers a standard function *ROOTS*. For our purposes, it must be either improved or replaced by a quite different and more accurate algorithm.

Time-domain analysis (impulse and step responses, forced responses to a given signal, etc.) can be performed by means of partial fraction expansion of circuit function and a subsequent inverse Laplace transformation of elementary terms. The first step is executable by a function *RESIDUE*. Numerically, the partial fraction expansion of a ratio of polynomials represents an ill-posed problem. If the denominator polynomial is near a polynomial with multiple roots, then small changes in the data, including roundoff errors, can make arbitrarily large changes in the resulting poles and residues [1]. On that account, the methods based on the poles representation are preferable. However, they are not included to the standard MATLAB functions.

Frequency responses can be readily obtained by a standard function *FREQS*. Analogously to the function *RESIDUE*, it starts from the system representation in a form of coefficients. It can be a source of significant errors in case of large systems and systems with a large spread of coefficients. For these events, we need a similar modified function which is based on the zero/pole system representation.

3 Proposed Conception

The new conception utilizes specially designed MATLAB functions *DEFL*, *FADDEJEV*, *LAGUER*, *ROOTSS*, *SRP*, *RESID*, and *FREQSZP* (see Fig. 2).

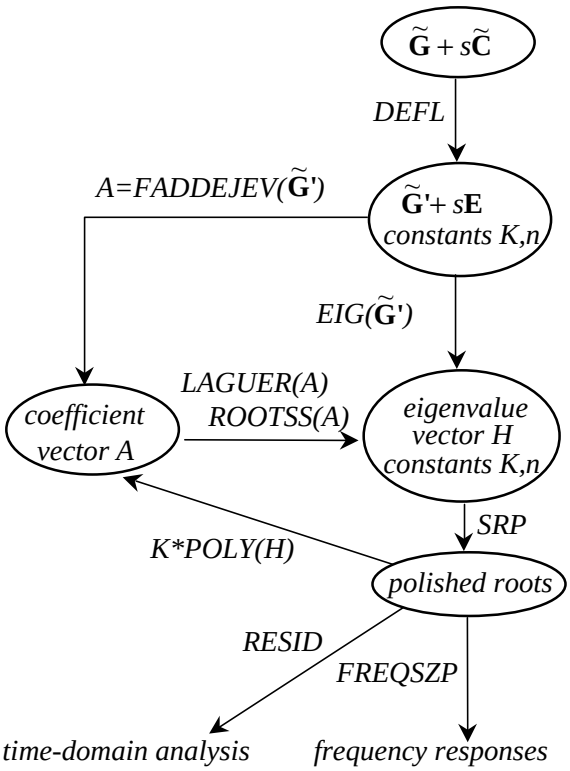

Figure 2. Proposed conception of semisymbolic and numerical analyses with the utilization of new MATLAB functions.

Function *DEFL* performs so-called deflation of a minor matrix $\tilde{\mathbf{G}} + s\tilde{\mathbf{C}}$. Deflation is a procedure of linear operations above the *NxN* minor matrix with the aim to transform it to another matrix $\tilde{\mathbf{G}}' + s\mathbf{E}$, where **E** is *nxn* unity matrix, $n \leq N$, and the following equality is held (K is a constant ensuring from the algorithm) [5]:

$$\det\left(\tilde{\mathbf{G}} + s\tilde{\mathbf{C}}\right) = K \det\left(\tilde{\mathbf{G}}' + s\mathbf{E}\right). \quad (5)$$

As results from (5), solution of generalized eigenvalue problem is transferred to the finding of eigenvalues of the matrix $\tilde{\mathbf{G}}'$. It is easily done using the standard function *EIG* (see Fig. 2).

Function *DEFL* has been adapted from the

algorithm of C.I.A. simulator [5]. It is called in a following form:

[G2,NN,KS,KL]=DEFL(N,C,G,ABSTOL,RELTOL)

where:

N	system dimension
C,G	input matrices of *NxN* dimension
ABSTOL	absolute tolerance (default value is 1e-20)
RELTOL	relative tolerance (default value is 1e-14)
G2	output transformed matrix of *NNxNN* dimension prepared for the standard function *EIG*
NN	dimension of reduced linear system
KS	sign of the constant *K*
KL	natural logarithm of constant *K*.

Depending of the sign +/- of constant *K*, the variable *KS* is set to 1/-1. In case of singular system, *KS* = 0. Natural logarithm of the absolute value of *K* is in the variable *KL*. Given method of *K* representation is necessary, because this constant gains the enormous values for large systems.

Transformation the generalized to standard eigenvalue problem enables direct computation of the system coefficients using Faddejev algorithm [6] without finding eigenvalues.

Of course, the coefficients can also be found from the eigenvalues using standard function *POLY*. Before this, eigenvalues have to be modified by the *SRP* (Secondary Root Polishing) function. Among others, this function ensures that all complex roots will have their exact complex conjugated counterparts [7]. Then the accuracy of computation coefficients by *POLY* function is increased dramatically.

The root finding from the coefficients is newly solved by two different methods. The first one starts from the standard MATLAB function ROOTS, which is modified with the utilization of MATLAB Symbolic Toolbox.

The original function *ROOTS* is based on the well-known method of companion matrix, which is created from the polynomial coefficients so that eigenvalues of the companion matrix are the same as the polynomial roots. The roots are then found after applying the function *EIG* to the matrix. The final part of the function *ROOTS* is as follows (here *c*, *a*, and *r* are polynomial vector, companion matrix, and the vector of eigenvalues):

```
% Polynomial roots via a companion matrix
n = length(c);
if n > 1
   a = diag(ones(1,n-2),-1);
   a(1,:) = -c(2:n) ./ c(1);
   r = [r;eig(a)];
end
```

The accuracy can be significantly increased using symbolic mode of the computation. If matrix *a* is defined as symbolic, then the function *EIG*(*a*) generates eigenvalues very precisely. The function *ROOTSS* is derived from the *ROOTS* using the following modification of its final part:

```
⋮
a(1,:) = -c(2:n) ./ c(1);
   a=sym(a);
   r = eig(a);
   r=double(r);
end
```

The function *ROOTSS* cannot be used for the analysis of large systems due to a time expensive computation. The second problem is that not all users have the symbolic toolbox. That is why the function *LAGUER* has been developed.

Function *LAGUER* is based on the Laguerr method of roots finding [3, 4]. This method belongs to the most accurate methods of finding roots of the polynomials with real, complex, and multiple roots. Along with the built-in root polishing algorithm [3], it offers better results than the method of companion matrix.

The original MATLAB function *RESIDUE* for partial fraction expansion was improved. The new function *RESID* includes a good algorithm by Chin and Steiglitz, which starts from the pole system representation [8].

The function *FREQSZP* has the same outputs as the standard *FREQS*, but zeros, poles, and constant *K* are the inputs instead of the numerator and denominator coefficients. This function helps to perform frequency analysis without numerical problems which can appear by the *FREQS* application to large systems.

4 EXAMPLE

Consider a cascade of three identical 2nd order single-input/single-output blocks with the transfer function

$$b_0 / \left(s^2 + b_1 s + b_0\right), b_0 = \omega_0^2, b_1 = \frac{\omega_0}{Q}, f_0 = 10kHz, Q = 80$$

Resulting system has three pairs of triple poles

-3.926990816987242e2 ±j6.283062587518110e4.

The following system MNA matrix (1) exists:

							1		
								1	
									1
	$-s-b_1$			1					
$-b_0$	b_0			s					
		$-s-b_1$			1				
	$-b_0$	b_0			s				
			$-s-b_1$			1			
		$-b_0$	b_0			s			

Poles will be computed from the minor matrix *Y1=G1+sC1* after omission first row and first column. We use standard function *EIG*:

```
H=-eig(G1,C1)
H =
-3.926990816987244e+002 +6.283062587518113e+004i
-3.926990816987243e+002 -6.283062587518111e+004i
-3.926990816987244e+002 -6.283062587518110e+004i
-3.926990816987242e+002 -6.283062587518109e+004i
-3.926990816987240e+002 +6.283062587518106e+004i
-3.926990816987240e+002 +6.283062587518105e+004i
NaN + NaNi
NaN + NaNi
NaN + NaNi
```

The reminder three poles are of "Not-a-Number" type. We exclude them from the vector H:

```
H=H(1:6);
```

It should be noted that slight inaccuracies have appeared in the result. This courses big errors in the system coefficients. Function A=poly(H) gives a wrong result (we cut numbers to 3 valid decimals)

[1 2.356e3 +j5.093e-11, 1.184e10 +j8.940e-8, 1.860e13 +j5e-1, 4.676e19 +j2.56e2, 3.672e22 +j4.026e8, 6.152e+28 -j1.374e11].

Now apply the new function *DEFL*:

```
[G2,NN,DS,DL]=defl(9,C1,G1);
H=-eig(G2)
H =
-3.926990816987242e+002 +6.283062587518110e+004i
-3.926990816987242e+002 -6.283062587518110e+004i
-3.926990816987242e+002 +6.283062587518110e+004i
-3.926990816987242e+002 -6.283062587518110e+004i
-3.926990816987242e+002 +6.283062587518110e+004i
-3.926990816987242e+002 -6.283062587518110e+004i
```

All eigenvalues are obtained very precisely. Applying A=poly(H) leads now to the correct result

[1, 2.356e3, 1.184e10, 1.860e13, 4.676e19, 3.672e22, 6.152e28].

Finally, let us demonstrate new functions of roots computation from the coefficients. First we generate coefficients which correspond to a tenfold root 1:

```
k=poly([1 1 1 1 1 1 1 1 1 1]);
```

The standard function *ROOTS* gives a result

```
1.039928012246194e+000 +1.435318401130502e-002i
1.039928012246194e+000 -1.435318401130502e-002i
1.021316655822595e+000 +3.578200663367344e-002i
1.021316655822595e+000 -3.578200663367344e-002i
9.940333131745804e-001 +3.980273373870628e-002i
9.940333131745804e-001 -3.980273373870628e-002i
9.713185238582574e-001 +2.560448876799779e-002i
9.713185238582574e-001 -2.560448876799779e-002i
9.839848398857208e-001
9.628221499110266e-001
```

Both new functions *ROOTSS* and *LAGUER* offer precisely ten times 1. However, function *LAGUER* is matchlessly faster.

Note: *Presented results relate to the MATLAB v.5.1.*

Acknowledgement: *This work is supported by the Grant Agency of the Czech Republic under grant No.* 102/97/0765.

REFERENCES

[1] D.Hanselman, B.Littlefield, "Mastering MATLAB 5. Prentice Hall, 1998.

[2] J.Vlach and K.Singhal, „Computer Methods for Circuit Analysis and Design," Van Nostrand Reinhold, N.Y., 1983.

[3] W.H.Press at al., "Numerical Recipes. The Art of Scientific Computing," Cambridge University Press.

[4] A.Ralston, "A First Course in Numerical Analysis," McGraw-Hill Company, 1965.

[5] J.Dobeš, "Analysis the Poles and Zeros for Expensive Analog and Digital Circuits,". Radioelektronika'99, Brno, Czech Republic, pp. 34-37.

[6] F.R.Gantmacher, „Teorija matric," Moskva, Nauka 1966.

[7] D.Biolek, „Secondary Root Polishing Techniques in Computer Circuit Analysis," Radioelektronika'99, Brno, Czech Republic, pp. 42-45.

[8] F.Y.Chin, K.Steiglitz, „An $0(A^2)$ Algorithm for Partial Fraction Expansion," IEEE Trans. on CAS, Vol.24, No.1, January 1977, pp.42-45.