Effective computations with 2-variable polynomial matrices in MATHEMATICA

P. Kujan*, M. Hromčík, M. Šebek*****
*Department of Control Engineering,
**Centre for Applied Cybernetics,
Faculty of Electrical Engineering,
Czech Technical University in Prague,
16627 Prague, Czech Republic
***Institute of Information Theory and Automation,
18208 Prague, Czech Republic

N.P. Karampetakis, E.N. Antoniou, S. Vologiannidis
Department of Mathematics,
Faculty of Science,
Aristotle University of Thessaloniki,
54006 Thessaloniki, Greece

Abstract— **This report describes our joint work on implementation of effective numerical routines for two-variable polynomial matrices in the *MATHEMATICA* software. New functions are connected to the already created program package, being developed at the Czech Technical University in Prague, which calculates only with polynomial matrices in one variable. As the first step, new polynomial object for 2-D polynomial matrix was created in the *MATHEMATICA* programming language. This was essential for good integration with the polynomial package mentioned above. Implementation of a bunch of efficient and reliable numerical routines followed.**

I. INTRODUCTION

In recent years the problematic of systems featuring parametric uncertainties has appeared in the center of many papers and books focused on robust control, see [1] for instant. One of the basic problems of this field is analysis of stability with respect to parameters that perform polynomially. Curing this case by new attractive numerical methods for polynomial matrices is the subject of this report.

More precisely, the task to be solved is defined as follows: Consider the 2-D scalar or matrix polynomial

$$P(s,q) = P_0(s) + qP_1(s) + q^2P_2(s) + \cdots + q^dP_d(s),$$

where

$$P_i(s) = P_{i0} + P_{i1}s + \cdots + P_{iN}s^N$$

$P(s,q)$ can be thought of as an uncertain polynomial in the variable s: for each value of q within a given region one obtains a fixed polynomial $P(s,.)$. This polynomial can become stable or unstable depending on the particular q [1]. It can be proved [1] that if $P(s,0)$ is stable and $P(s,q)$ has invariant degree for all q real, then there exist such numbers $q_{min} \leq 0$, $q_{max} \geq 0$ that $P(s,q)$ remains stable for all $q \in \langle q_{min}, q_{max}\rangle$. Finding such bounds on q via a fast and numerically reliable procedure is exactly the goal of this contribution.

[1]We consider the *Hurwitz* stability here, i.e. a polynomial $p(s)$ is said to be stable if all its roots lie in the strict complex left half plane.

II. THEORETICAL SOLUTION AND NUMERICAL ASPECTS

The solution described in (Barmish, 1994) relies on the Hurwitz stability matrix and its properties.

A. Scalar Case

Let us consider the scalar case first. It can be shown (Barmish, 1994) that the Hurwitz matrix $H(P)$ related to the polynomial $P(s,.)$ can be expressed for each q as

$$H_P = H_{P_0} + qH_{P_1} + \cdots + q^dH_{P_d}$$

and that it has full rank unless some roots of $P(s,q_i)$ lie on the imaginary axis. If the invariant degree of $P(s,q)$ is preserved for all q, then the motion of the roots of $P(s,.)$ depends continuously on the changes of the parameter q and the desired values q_{min}, q_{max} can be found among the set $r = \{q_1, q_2, \ldots\}$ of roots of $\det(H(q))$. As $P(s,0)$ is stable by assumption, all the roots are nonzero. q_{min} equals the negative root smallest in absolute value whereas q_{max} is the largest nonnegative root. If the roots of $H(q)$ are all positive, $q_{min} = -\infty$. And on the contrary, if all q_i are negative, $q_{max} = \infty$.

In the literature, the computation of these values is typically transformed to the generalized eigenvalues problem for the companion matrix related to $H(q)$ (Barmish, 1994). However, a new algorithm for the polynomial matrix determinant based on FFT has been developed recently and has proved to be numerically reliable and very fast (Hromčík and Šebek, 1999) and can be proposed to be utilized here:

Algorithm 1: Determinant of Polynomial Matrix

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

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

Step 1 *(Result's degree estimation)*
Compute the upper bound for the degree of the determinant:

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

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 *(Evaluation of the input matrix at the Fourier points s_i)*
Perform direct FFT 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 *(Constant matrix manipulation)*
Compute the vector $\boldsymbol{z} = [z_0, z_1, \ldots, z_{N_{det}}]$, where $z_k = \det(Y_k), k = 0, 1, \ldots, N_{det}$. If $P(s)$ is real, evaluate the determinants $z_k = \det(Y_k)$ only for $k = 0, 1, \ldots, \lceil N_{det}/2 \rceil$. The remaining values z_i can be assigned according to Theorem 2.2 as $z_{N_{det}+1-k} = \det(Y_k^*) = z_k^*$.

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

Using this algorithm to evaluate the determinant of $H_P(q)$ and employing a traditional routine for subsequent computation of its roots [5], instead of the traditional procedure with generalized eigenvalues computation, helps to avoid numerical difficulties during evaluation of large size problems on a computer and also decreases the computational time considerably [6].

B. MIMO Case

The ideas of the introduced algorithm can be easily extended to cover also the evaluation of 2-D polynomial matrix determinant - formally, just the (one-dimensional) FFT's in the first and third step of the algorithm above are replaced by 2-D FFT's [4], and indexing in the *Step 2* becomes two dimensional as well.

Algorithm 2: Determinant of 2-D Polynomial Matrix

Input: Square 2-D polynomial matrix $P(s,t)$ of size n and degrees d_s and d_t.

Output: 2-D scalar polynomial $p(s,t)$ - the determinant of $P(s,t)$.

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

$$d_{det_s} = n \cdot d_s, d_{det_t} = n \cdot d_t$$

Step 2 Using *FFT* algorithm, perform direct 2-D DFT at $(d_{det_s}+1) \times (d_{det_t}+1)$ points on the set $\mathcal{P} = \{P_{m,n}\}$ of coefficient matrices of $P(s,t)$ and obtain the set $\mathcal{Y} = \{Y_{k,l} | k = 0, 1, \ldots, d_{det_s}, l = 0, 1, \ldots, d_{det_t}\}$.

Step 3 Compute the set $\boldsymbol{z} = \{z_{k,l} | k = 0, 1, \ldots, d_{det_s}, l = 0, 1, \ldots, d_{det_t}\}$, where $z_{k,l} = \det(Y_{k,l})$

Step 4 Perform inverse 2-D DFT on $\boldsymbol{z}$ using the *FFT* algorithm to obtain the coefficient set $\boldsymbol{p} = \{p_{m,n} | m = 0, 1, \ldots, d_{det_s}, n = 0, 1, \ldots, d_{det_t}\}$ of the determinant

$$p(s,t) = \sum_{m=o}^{d_{det_s}} \sum_{n=o}^{d_{det_t}} p_{m,n} s^m t^n$$

◇

Combining both the methods for one- and two-dimensional polynomial determinants can be used to treat the multi-variable case of the problem: if $P(s,q)$ is a matrix of uncertain polynomials, computing its 2-D polynomial determinant $p(s,q) = \det(P(s,q)) = p_0(s) + q p_1(s) + \cdots + q^D p_D(s))$ reduces the matrix problem to the scalar case which in turn can be efficiently solved via 1-D polynomial determinant computation using the algorithm listed above.

III. New polynomial objects for 2-D polynomial matrices in *Mathematica*

New objects are connected to the already created program package [2] , which calculates only with polynomial matrices in one variable.

Similarly as in [2], also here the new functions were created which are used to input 2-D polynomial matrices. New polynomial objects join a polynomial matrix or a polynomial with their variables. 2-D polynomial matrixes could be entered in three fundamental forms using the following functions:

- **Polynomial Matrix Form**
The command `PM[pm2d,{var1,var2}]` (Polynomial Matrix) creates a new polynomial matrix object. Input arguments are a rectangular matrix (list) *pm2d* with polynomial entries (in standard *MATHEMATICA* notation or in scalar polynomial object form) and its considered variables $\{var1,var2\}$.
The function `PM[]` for 2-D polynomial matrices is similar 1-D polynomial matrices. The only difference is a statement of the list with two variable instead of one.
Example 1. 2-D polynomial matrix
$$\begin{pmatrix} 1+2\,s\,z & 3\,s^2+4\,z \\ 5+7\,s^2+6\,z & 8\,s^2\,z \end{pmatrix}$$
in `PM[]` representation.
In[1] :=
```
PM[{{1 + 2s z, 3 s^2 + 4z},
    {5 + 6z + 7s^2, 8s^2 z}}, {s, z}]
```
Out[1] =
$$\begin{pmatrix} 1+2\,s\,z & 3\,s^2+4\,z \\ 5+7\,s^2+6\,z & 8\,s^2\,z \end{pmatrix}_{\{s,z\}}$$

- **Polynomial Matrix Coefficients I.**
The command `PMC[pmc2d,{var1,var2}]` (Polynomial Matrix Coefficients) creates a new polynomial matrix object. Input arguments are a list of lists of rectangular constant matrix with numbers or symbols *pmc2d*. The second argument are related variables $\{var1,var2\}$.
Example 2. 2-D polynomial matrix from example 1 in `PMC[]` representation.
In[2] :=

```
PMC[{
  {{{1,0},{5,0}},{{0,4},{6,0}}},
  {{{0,0},{0,0}},{{2,0},{0,0}}},
  {{{0,3},{7,0}},{{0,0},{0,8}}}
}, {s,z}];
```

- **Polynomial Matrix Coefficients II.**
  Can be used the command `PMC[{pm1, pm2, ...}, var1]`, where as first argument is placed list with 1-D polynomial objects $\{pm1, pm2, \ldots\}$ in variable *var2* and the second argument is variable *var1*.
  In[3] :=
  ```
  PMC[{
    PM[{{1, 4z},{5 + 6z, 0}}],
    PM[{{2z, 0},{0, 0}}]
    PM[{{0, 3},{7, 8z}}]
   }, s]
  ```

2-D scalar polynomials can be set in two forms similarly:

- **Scalar Polynomial**
  The command `P[pol2d, {var1,var2}]` (Polynomial) creates a new scalar polynomial object. Input arguments are a 2-D scalar polynomial *pol2d* in standard *MATHEMATICA* notation and its variables $\{var1,var2\}$.
  Example 3. 2-D scalar polynomial
  $$1 + 2sz + 3s^2z + 4z^2$$
  in `P[]` representation.
  In[4] :=
  ```
  P[1 + 2s z + 3 s^2 z + 4 z^2, {s,z}]
  ```
  Out[4] =
  $$\left(1 + 2\,s z + 3\,s^2 z + 4\,z^2\right)_{\{s,z\}}$$
- **Polynomial Coefficients I.**
  The command `PC[pol2d, {var1,var2}]` (Polynomial Coefficients) creates a new scalar polynomial object. Input arguments are a list of lists of scalar polynomial coefficients and its considered variables $\{var1,var2\}$.
  Example 4. Scalar polynomial from Example 3 in `PC[]` representation
  In[5] :=
  ```
  PC[{{1,0,4}, {0,2,0}, {0,3,0}},
      {s, z}];
  ```
- **Polynomial Coefficients II.**
  Analogously to polynomial matrix coefficients it is possible to use by the command `PMC[{p1, p2, ...}, var1]`, where first argument `{p1, p2, ...}` is list with 1-D scalar polynomial objects in variable *var2* and the second argument is variable *var1*.
  Example 5.
  In[6] :=
  ```
  PC[{
    P[{1+4z^2}],
    P[{2z}]
    P[{3z}]
   }, s]
  ```

The conversion of 1-D polynomial matrix with one linear parameter to 2-D polynomial matrix in which the given specific parameter represents the second variable is easy.

In[7] :=

```
a = PM[{{1 + 3q^2 + 4s, 5q + 6s^2},
        {7 + 8s, 9q^2}}, s];
a2d = PM[a, {s,q}]
```

Out[7] :=

$$\begin{pmatrix} 1+3\,q^2+4\,s & 5\,q+6\,s^2 \\ 7+8\,s & 9\,q^2 \end{pmatrix}_s$$

Out[8] :=

$$\begin{pmatrix} 1+3\,q^2+4\,s & 5\,q+6\,s^2 \\ 7+8\,s & 9\,q^2 \end{pmatrix}_{\{s,q\}}$$

A. Implementation in MATHEMATICA

Function `PM[]` returns the 2-D polynomial matrix object which is represented by the function `PolyMat[pmc2d,{var1,var2},{deg1,deg2}]`.
If input 2-D polynomial matrix is

$$P = P_0(s) + qP_1(s) + \cdots + q^{d_q}P_{d_q},$$

where $P_0(s) = P_0^0 + P_1^0 s + \ldots + P_{d_{s_0}}^0 s^{d_{s_0}}$ and $P_1(s) = P_0^1 + P_1^1 s + \ldots + P_{d_{s_1}}^1 s^{d_{s_1}}$, ..., then *pmc2d* is list of lists of scalar matrices in this form

$$\{\{P_0^0, P_1^0, \ldots, P_{d_s}^0\}, \{P_0^1, P_1^1, \ldots, P_{d_s}^1\}, \ldots$$
$$, \{P_0^{d_q}, P_1^{d_q}, \ldots, P_{d_s}^{d_q}\}\},$$

where $d_s = \max\{d_{s_0}, d_{s_1}, \ldots, d_{s_{d_q}}\}$.

Preview of MATHEMATICA code for function `PM[]` which create 2-D polynomial matrix object.

```
PM[pm_?MatrixQ, {var1_Symbol, var2_Symbol}]
         /; PMTest2D[pm, {var1, var2}]:=
Module[{coef, dimcoef, maxrc, plc},
  coef = Map[
         CoefficientList[#, {var1, var2}]&
         ,pm, {2}
        ];
  dimcoef = Map[Dimensions, coef, {2}];
  maxrc = Max/@Transpose[Flatten[dimcoef,1]];
  plc = Map[ExpandDim[#,maxrc]&, coef, {2}];
  PolyMat[
    Transpose[plc, {3, 4, 1, 2}],
     {var1, var2}, maxrc - 1]
];
```

Function `PMC[]` has the identical output as `PM[]`, it's function `PolyMat[]`.

Preview of MATHEMATICA code for definition function `PMC[]` (in variant II.) which crate 2-D polynomial matrix object.

```
PMC[
 pms:{PolyMat[_, var2_Symbol, TypeDeg]...},
 var1_Symbol] /; PMCTest2D1[pms, var1]:=
Module[
  {degs = Deg /@ pms, maxdeg2, repms, r,
   c, adeg},
  If[Not[SameQ @@ (Size /@ pms)],
    Message[pmc2d:errsize]; Return[Null],
    {r, c} = Size[First[pms]]];
    maxdeg2 = Max[degs];
    PolyMat[
      If[Not[SameQ @@ degs],
        If[(adeg = Deg[#]) < maxdeg2,
          Join[#[[1]],
            PrepMat[r,c,maxdeg2-adeg,0]
          ],
          #[[1]]
        ]& /@ pms,
        #[[1]]& /@ pms
      ]
      , {var1, var2}
      , {Length[pms] - 1, maxdeg2}
    ]
];
```

B. Implementation of 2-D polynomial matrix determinant

The routines implemented in MATHEMATICA for our new 2-D polynomial objects include the determinat computation. For the polynomial matrix with numerical coefficients the implemented algorithm is adopted from [3] and is described in Algorithm 2. It is the most efficient published method for polynomial matrix determinant computation. Non-numerical polynomial matrices are resolved by standard *MATHEMATICA* function for symbolic determinant.

Example 6.

In[8] := `A=PM[`$\begin{pmatrix} 2.3+8.1\,s^2-5.7\,s\,z^2 & -5.6+7.4\,s\,z \\ 9.3\,s+5.2\,z & -7.4\,z^2 \end{pmatrix}$`];`

`Det[A]`

Out[8] :=

$$(\,52.08\,s+29.12\,z-68.82\,s^2\,z+\ldots+42.18\,s\,z^4\,)_{\{s,z\}}$$

C. Experimental testing

As expected, the implemented methods specifically tailored for 2-D polynomial objects are considerably faster than the general symbolic routines incorporated in *MATHEMATICA*.

A star (*) in tables means that the execution time exceed 2000 seconds.

TABLE I

TIME - CONSUMPTION. SQUARE TEST MATRICES OF SIZE n AND DEGREE $\{d,d\}$ HAVE RANDOM REAL COEFFICIENTS.

Input sq. mat. n	$\{d,d\}$	Poly `Det[A]`	Standard `Map[Expand,Det[sA],{2}]`
3	2	0.03	0.01
5	3	0.06	0.8
7	5	0.4	78.8
9	6	1.7	1590.3
10	8	4.53	*

IV. PRACTICAL EXAMPLE: UNCERTAIN SYSTEM ANALYSIS

The following *MATHEMATICA* session illustrates the use of our polynomial package with its 2-D determinat solver.

Consider an uncertain polynomial matrix $P(s,q) = P_0(s) + qP_1(s) + q^2P_2(s)$ where

In[9] :=

`P0 = PM[`$\begin{pmatrix} 5+2\,s+5\,s^2 & s \\ 4\,s+3\,s^2 & 5 \end{pmatrix}$`];`

`P1 = PM[`$\begin{pmatrix} 1 & 1+s+2\,s^2 \\ 8 & 6+6\,s \end{pmatrix}$`];`

`P2 = PM[`$\begin{pmatrix} 4 & 0 \\ 7+6\,s+6\,s^2 & 5+5\,s+9\,s^2 \end{pmatrix}$`];`

Direct test says that for $q=0$ the matrix $P(s,0) = P_0(s)$ is stable. The desired bounds $q_{min} \le 0, q_{max} \ge 0$ such that $P(s,q)$ remains Hurwitz stable for $q_{min} < q < q_{max}$ with respect to s can be achieved according to the considerations in section II.

First the determinant of $P(s,q)$ is computed.

In[10] :=

```
Psq = PM[P0+P1*P[q,s]+P2*P[q^2,s], {q,s}]
(* Psq = PM[{P0,P1,P2}, q]] *)
p = Det[Psq]
```

Out[10] =

$$\begin{pmatrix} 5+q+\ldots+5\,s^2 & q+\ldots+2\,q\,s^2 \\ 8\,q+7\,q^2+\ldots+6\,q^2\,s^2 & 5+6\,q+\ldots+9\,q^2\,s^2 \end{pmatrix}_{\{q,s\}}$$

Out[11] =

$$(\,25.+35.\,q+43.\,q^2+\ldots+45.\,q^2\,s^4-12.\,q^3\,s^4\,)_{\{q,s\}}$$

The Hurwitz matrix related to this uncertain polynomial equals

In[11]:=

```
Hq = Hurwitz[p,q]
```

Out[11] =

$$\begin{pmatrix} -3+\ldots+18\,s^3 & 10+\ldots & 0 & 0 \\ -6\,s+\ldots+12\,s^3 & 21+\ldots & 25+\ldots & 0 \\ 0 & -3+\ldots & 10+\ldots & 0 \\ 0 & -6\,s+\ldots & 21+\ldots & 25+\ldots \end{pmatrix}_q$$

Finite zeros of $P(q)$ with respect to q are the roots of its determinant:

In[12]:=

```
hq = Det[Hq]
```

Out[12]=

$$(-21000. + 75000.\, s + \ldots + 73000.\, s^{11} + 18000.\, s^{12})_q$$

and its roots equal

In[13]:=

```
roots = Roots[hq]
{qmin, qmax} =
  {-Abs[Min[Select[roots, Negative]]],
    Min[Select[roots, Positive]]}
```

Out[13] =

$$\{-1.4, -0.51 - 0.02\, I, \ldots, 0.12, 0.75, 7.43\}\}$$

Out[14]=

$$\{-1.4, 0.125\}$$

Eventually, q_{min} and q_{max} follow directly:

$$q_{min} = -1.4,\ q_{max} = 0.125.$$

V. Conclusion

The 2-D polynomial derminat solver of the polynomial package for *MATHEMATICA*, being developed at the Czech Technical University in Prague, has been presented. The algorithm used was explained, the implementation issue have been discussed, and finally, the routine was used to solve a robust control analysis problem.

Acknowledgment

The work of P.Kujan has been supported by the Ministry of Education of the Czech Republic under contract No. INGO LA156. The work of M. Hromčík has been supported by the Ministry of Education of the Czech Republic under contract No. LN00B096. The work of M. Šebek has been supported by the Ministry of Education of the Czech Republic under contract KONTAKT No. ME496.

References

[1] B.R. Barmish, *New Tools for Robustness of Linear Systems*, Macmillan Inc., New York, 1999.

[2] P. Kujan, M. Hromčík, M. Šebek. *New package for effective polynomial computation in Mathematica.* The 11 th Mediterranean Conference on Control and Automation (MED03), Rhodes, Greece, June 18-20, 2003.

[3] M. Hromčík, M. Šebek. *Fast Fourier Transform and Robustness Analysis with Respect to Parametric Uncertainties.* Preprints of the IFAC Robust Control Design Conference - ROCOND, (Prague, CZ), June 2000.

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

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

[6] M. Hromčík, M. Šebek, *New Algorithm for Polynomial Matrix Determinant Based on FFT, Proceedings of the European Control Conference ECC'99*, Karlsruhe, Germany, 1999.

[7] T.B. Bahner, MATHEMATICA *for Scientists and Engineers*, Addison-Wesley Publishing Company, Inc., 1995.

[8] D.B. Wagner, *Power programming with* MATHEMATICA*: the Kernel*, McGraw - Hill, 1996.