数理解析研究所講究録
1085 巻 1999 年 99-107

The CHACM Method for Computing the Characteristic Polynomial of a Polynomial Matrix

Bo Yu

Department of Mathematics, Jilin University, China *

北本 卓也 (Takuya KITAMOTO)

Institute of Mathematics, University of Tsukuba, Japan†

1 Introduction

The characteristic polynomial of a square matrix is a basic concept in linear algebra and its computation has important applications in automatic control and other fields. For a constant martrix, several algorithms for computing the characteristic polynomial has been given, and the major ones are Faddeev-Leverrier's method, Lagrange interpolation method, Hessenberg methods and etc. (see [1-2, 4-12]). Some times, we need to compute the characteristic polynomial of a polynomial matrix or exactly compute the characteristic polynomial of a constant matrix with integer elements. As being a fraction-free method, Faddeev-Leverrier's method sketched below is the only one among above mentioned methods which can be applied to such problems directly.

Faddeev-Leverrier's method ([2, 4]): Let

$$\begin{aligned} A_1 &= A, \\ c_i &= -\frac{\operatorname{trace} A_i}{i} \quad (1 \le i \le n), \\ A_{i+1} &= A(A_i + c_i I) \quad (1 \le i \le n-1), \end{aligned}$$

then the characteristic polynomial of matrix A is given by $c(\lambda) = \lambda^n + c_1\lambda^{n-1} + \cdots + c_n$. This is a fraction-free method and hence can also be applied to a polynomial matrix. It needs $n^3(n-1)$ polynomial multiplications when it is used to compute the characteristic polynomial of a polynomial matrix. Because the multiplication of two polynomials with few terms and the multiplication of two polynomials with many terms are very different

*yubo@math.tsukuba.ac.jp

†kita@math.tsukuba.ac.jp

in computational time, it would be better to count the number of multiplications between numbers than to count the numeber of multiplications between polynomials. For an $n \times n$ univariate polynomial matrix $A(x) = A_0 + A_1 x + \cdots + A_d x^d$ with dense matrices A_i's, Faddeev-Leverrier's method needs

$$\sum_{i=1}^{n-1} (d+1)(id+1) = \frac{n^2 d^2}{2} + l.d.t.$$

matrix mulitiplications, i.e., $\frac{n^5 d^2}{2} + l.d.t.$ number multiplications. Here, by *l.d.t.*, we mean "lower degree terms in $n$ and $d$".

Recently, Kitamoto presented in [5] a new algorithm for computing the characteristc polynomial of a polynomial matrix based on Cayley-Hamilton theorem. We refer the method given in [5] as CHTB method. The main idea of CHTB method is as follows:

The CHTB method ([5]): Let $A(x) = A_0 + A_1 x + \cdots + A_d x^d$ be an $n \times n$ polynomial matrix. Compute first the eigenvalues $\lambda_1, \ldots, \lambda_n$ and eigenvectors $u_1, \ldots, u_n$ of A_0 by some numerical method. If $\lambda_i = \lambda_j$ for any $i \neq j$, then by a similarity transformation, say $\tilde{A}(x) = S^{-1} A(x) S$ where $S$ can be constructed from $u_1, \ldots, u_n$, if needed, there must be a $1 \leq l \leq n$ such that the following matrix

$$G = \left([I]_l, [\tilde{A}_0]_l, \ldots, [\tilde{A}_0^{n-1}]_l \right) \tag{1.1}$$

(where, for a matrix $M$, $[M]_l$ denotes the lth column of $M$) is nonsingular. Let

$$c(x, \lambda) = \lambda^n + c_1(x)\lambda^{n-1} + \cdots + c_n(x),$$

with $c_i(x) = c_{i0} + c_{i1}x + \cdots + c_{ik_i} x^{k_i}$, be the characteristic polynomial of $A(x)$. By Cayley-Hamilton theorem, and noticing that $\tilde{A}(x)$ has the same characteristic polynomial with $A(x)$, we have

$$c(x, \tilde{A}(x)) \equiv 0,$$

and hence

$$[c(x, \tilde{A}(x))]_l \equiv 0,$$

i.e.,

$$\begin{aligned} &[\tilde{A}(x)^n]_l + (c_{10} + c_{11}x + \cdots + c_{1k_1}x^{k_1})[\tilde{A}(x)^{n-1}]_l \\ &+ \cdots + (c_{n0} + c_{n1}x + \cdots + c_{nk_n}x^{k_n})[I]_l \equiv 0. \end{aligned} \tag{1.2}$$

By equating the coefficients of x^i $(i = 0, \ldots, D = nd)$ in (1.2), we get

$$Gh_0 = -[\tilde{A}_0^n]_l, \tag{1.3}$$

$$Gh_i = -f_i, \quad i = 1, \ldots, D, \tag{1.4}$$

$h_i = (c_{1i}, \ldots, c_{ni})^{\mathrm{T}}, (i = 0, \ldots, D)$, f_i is the coefficient of x^i in

$$[\tilde{A}_0^n]_l + \left([I]_l, [\tilde{A}(x)]_l, \ldots, [\tilde{A}(x)^{n-1}]_l \right) \begin{bmatrix} c_{n0} + \cdots + c_{n(i-1)}x^{i-1} \\ \vdots \\ c_{10} + \cdots + c_{1(i-1)}x^{i-1} \end{bmatrix}.$$

Solving linear systrems in (1.3), (1.4), we get c_{ij}.

This method needs $(d+1)n^3$ polynomial multiplications. It is also observed that this method is faster than Faddeev-Leverrier's method even when $d+1$ is bigger than n. The shortcoming of the CHTB method is that it cann't be used to a polynomial matrix that A_0 has multiple eigenvalues and, it needs to compute first the eigenvalues and eigenvectors of A_0.

In this paper, by introducing in an artificial constant matrix and an auxiliary variable, we give a novel method, called Cayley-Hamilton artificial constant matrix method (CHACM method). It needs no condition on the given matrix, needs not to solve any eigenvalue problem and is division-free. The amount of the computation is asympototically $\frac{7}{12}$ of that of Faddeev-Leverrier's method.

In Section 2, the main idea of CHACM method are formulated and the CHACM algorithm with an improved version are given. In Section 3, computational tests are given to show that our algorithms really work and to compare the CPU-time of our methods with Faddeev-Leverrier's method and other methods.

2 The CHACM method

Let

$$E = \begin{pmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{pmatrix}.$$

E satisfies that

$$([I]_1, [E]_1, \ldots, [E^{n-1}]_1) = I,$$

and hence is the best matrix for serving as the constant matrix in Cayley-Hamilton theorem based methods. We will use it as the artificial constant matrix to give a novel method for computing characteristic polynomial.

To compute the characteristic polynomial of an $n \times n$ matrix A with elements in, e.g., $Q[z_1, \ldots, z_m]$, we construct $M(x) = E + x(A - E) = E + xB$. If $c(x, \lambda)$ is the characteristic polynomial of $M(x)$, then $c(\lambda) = c(1, \lambda)$ is the characteristic polynomial of A. We give an algorithm for computing $c(x, \lambda)$ and therefore an algorithm for computing the characteristic polynomial of the matrix A.

Because E is not only the constant matrix of $M(x)$ as the polynomial matrix with variates $x, z_1, \ldots, z_1$, but also the constant matrix of $M(x)$ as the polynomial matrix with one variate x, we will treat $z_1, \ldots, z_1$ as paramteric coefficients. This will make the program simpler and faster.

Let

$$c(x, \lambda) = \lambda^n + c_1(x)\lambda^{n-1} + \cdots + c_n(x).$$

Because $c_i(x)$ is the sum of all i-th order principal minors of $M(x)$, the degree of $c_i(x)$ in x is i. Let $c_i(x) = c_{i0} + c_{i1}x + \cdots + c_{ii}x^i$. By Cayley-Hamilton theorem,

$$c(x, M(x)) \equiv 0,$$

and thus $[c(x, M(x))]_1 \equiv 0$, i.e.,

$$[M(x)^n]_1 + c_1(x)\left[M(x)^{n-1}\right]_1 + \cdots + c_n(x)\left[M(x)^0\right]_1 \equiv 0. \tag{2.1}$$

Let $[M(x)^i]_1 = m_{i0} + m_{i1}x + \cdots + m_{ii}x^i$. It is easy to see that:

(1) $(m_{00}, m_{10}, \ldots, m_{(n-1)0}) = I$;

(2) $m_{n0} = 0$;

(3) If $m_{(i-1)1}, \ldots, m_{(i-1)(i-1)}$ have been computed, then by the ralation $M(x)^i = (E + xB)M(x)^{i-1}$, m_{i1}, ..., m_{ii} can be computed by

$$m_{ij} = E \cdot m_{(i-1)j} + B \cdot m_{(i-1)(j-1)}, \quad (1 \leq j \leq i),$$

here we set $m_{(i-1)i} = 0$. The computation of m_{ij}'s from $m_{(i-1)j}$'s need $i - 1$ multiplications of vectors by matrices and, the computation of all m_{ij}'s need $\frac{1}{2}n^4 + O(n^3)$ multiplications in D.

Substituting $[M(x)^i]_1$ and $c_i(x)$ by their representations, (2.1) can be writen out as follows

$$\begin{aligned}&(m_{n1}x + \cdots + m_{nn}x^n)+\\&(c_{11}x)(m_{(n-1)10} + m_{(n-1)1}x + \cdots + m_{(n-1)(n-1)}x^{n-1})\\&+ \cdots + (c_{n1}x + \cdots + c_{nn}x^n)(m_{00}) \equiv 0\end{aligned} \tag{2.2}$$

By equating the constant terms in the two sides of (2.2), we get

$$\begin{pmatrix} c_{n0} \\ \vdots \\ c_{10} \end{pmatrix} = -m_{n0} = 0. \tag{2.3}$$

By equating the coefficients of x in the two sides of (2.2), from(2.3), we get

$$\begin{pmatrix} c_{n1} \\ \vdots \\ c_{11} \end{pmatrix} = -m_{n1}. \tag{2.4}$$

Generally, for $2 \leq k \leq n$, by equating the coefficients of x^k in the two sides of (2.2), we get

$$\begin{pmatrix} c_{nk} \\ \vdots \\ c_{kk} \end{pmatrix} = -\left[m_{nk} + \sum_{1\leq i\leq n} \sum_{1\leq j\leq \min\{i,k-1\}} c_{ij}m_{(n-i)(k-j)}\right]_1^{n-k+1}, \tag{2.5}$$

where, for a vector $u = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}$, $[u]_i^j = \begin{pmatrix} u_i \\ \vdots \\ u_j \end{pmatrix}$.

For $2 \leq i \leq n$, computation of c_{ij} $(i \leq j \leq n)$ need $(i-1)(n-i+1)^2$ multiplications in D and, computation of all c_{ij}'s need

$$1 \cdot (n-1)^2 + 2 \cdot (n-2)^2 + \cdots + (n-2) \cdot 2^2 + (n-1) \cdot 1 = \frac{n^4}{12} - \frac{n^2}{12},$$

Let $c_i = \sum\limits_{j=1}^{i} c_{ij}$, then $c(\lambda) = \lambda^n + c_1\lambda^{n-1} + \cdots + c_n$ gives the characteristic polynomial of A.

Totally, it needs $\frac{7}{12}n^4 + O(n^3)$ multiplications in D.

By above discussion, we can give the Cayley-Hamilton artificial constant matrix algorithm, or, in abbreviation, CHACM algorithm, for computing characteristic polynomial in Algorithm 2.1.

Algorithm 2.1 (CHACM algorithm)

```
Input: A square polynomial matrix $A$.
Output: The characteristic polynomial of $A$.
Step 1: $A = A - E$.
Step 2: Computing $m_{ij}$'s.
    for $i = 1$ to $n$ do
        $m_{(i-1)0} = e_i$
    end
    $m_{n0} = 0$
    for $i = 1$ to $n$ do
        for $j = 1$ to $i$ do
            $m_{ij} = Em_{(i-1)j} + Am_{(i-1)(j-1)}$
        end
    end
Step 3: Computing $c_{ij}$'s.
    for $k = 1$ to $n$ do
        for $i = 1$ to $n$ do
            for $j = 1$ to $\min\{i, k-1\}$ do
                $[m_{nk}]_1^{n-k+1} = [m_{nk}]_1^{n-k+1} + [m_{nj}]_{n-i+1}[m_{(n-i)(k-j)}]_1^{n-k+1}$
            end
        end
    end
Step 4: Computing $c_i$'s.
```

```
for $i = 1$ to $n$ do
    $c_i = 0$
    for $j = 1$ to $i$ do
        $c_i = c_i - [m_{nj}]_{n-i+1}$
    end
end
```

Notice that, we can also compute the vectors

$$\begin{array}{l} m_{00} \\ m_{10}, m_{11} \\ \cdots\cdots\cdots \\ m_{n0}, m_{n1}, \cdots, m_{nn} \end{array}$$

column by column, and that if the first $j+1$ columns of vectors in above table have been computed then $c_{jj}, \ldots, c_{nj}$ can be computed from them. Once a new c_{ij} is computed, we can add the productions

$$c_{ij} m_{(n-i)k}, \quad k = 1, \ldots, \min\{j, n-i\}$$

to $m_{n(j+k)}$. And, once a new vector m_{ij} is computed, we can add the products

$$c_{(n-i)k} m_{ij}, \quad k = 1, \ldots, \min\{j-1, n-i\}$$

to $m_{n(j+k)}$ too. As the result, when above procedure is finished, the first $n-k+1$ elements of m_{nk} $(k = 1, \ldots, n)$ is that $-(c_{nk}, \ldots, c_{kk})^{\mathrm{T}}$. By this observation, we give the improved version of the CHACM algorithm, which compute m_{ij}'s and c_{ij}'s is one loop, in Algorithm 2.2.

Algorithm 2.2 (Improved version of the CHACM algorithm)

Input: A square polynomial matrix A.

Output: The characteristic polynomial of A.

Step 1: $A = A - E$

Step 2: Computing m_{ij}'s and c_{ij}'s.

```
for $i = 1$ to $n$ do
    $m_{(i-1)0} = e_i$
end
$m_{n0} = 0$
for $j = 1$ to $n$ do
    $m_{jj} = Am_{(j-1)(j-1)}$
    for $k = 1$ to $\min\{n-j, j-1\}$ do
      $[m_{n(j+k)}]_1^{n-j-k+1} = [m_{n(j+k)}]_1^{n-j-k+1} + [m_{nk}]_{j+1}[m_{jj}]_1^{n-j-k+1}$
    end
    for $i = j+1$ to $n-1$ do
      $m_{ij} = Em_{(i-1)j} + Am_{(i-1)(j-1)}$
      for $k = 1$ to $\min\{n-i, j-1\}$ do
        $[m_{n(j+k)}]_1^{n-j-k+1} = [m_{n(j+k)}]_1^{n-j-k+1} + [m_{nk}]_{i+1}[m_{ij}]_{n-j-k+1}$
      end
    end
    $[m_{nj}]_1^{n-j+1} = [m_{nj}]_1^{n-j+1} + [m_{nj} + Em_{(n-1)j} + Am_{(n-1)(j-1)}]_1^{n-j+1}$
    for $i = j$ to $n$ do
      for $k = 1$ to $\min\{n-i, j\}$ do
        $[m_{n(j+k)}]_1^{n-j-k+1} = [m_{n(j+k)}]_1^{n-j-k+1} + [m_{nj}]_{n-i+1}[m_{(n-i)k}]_1^{n-j-k+1}$
      end
    end
  end
```

Step 3: Computing c_i's.

```
  for $i = 1$ to $n$ do
    $c_i = 0$
    for $j = 1$ to $i$ do
      $c_i = c_i - [m_{nj}]_{n-i+1}$
    end
  end
```

3 Computational tests

In this section, we give some computational results. The algorithms are programed in *Mathematica.* The computational results show the correctness of our algorithms (at the present stage, we have only programmed the Algorithm 2.1.) and compare the CPU-time of our algorithms with that of Faddeev-Leverrier's method and internal function of *Mathematica* for computing the characteristic polynomial. We also compare our algorithm with Danilevakii's method (see [2]), of which a fraction-free modification has been given by Moritsugu and Kuriyama in [6] (English translation, [7]) (test results in [7] showed that for univariate polynomial matrices of dimension 3 to 20 the fraction-free version can reduce the CPU-time by about 70% at most from the original Danilevskii's method).

Using *Mathematica* Ver. 3.0 on Pentium II 233MHz, 128Mbyte RAM, we have computed the characteristic polynomial of some univariate polynomial matrices of degree 1, 2 and 3 and various dimensions with randomly generated integer coefficients. The following tables show the comparation of CPU-time by Algorithm 2.1 (**CHACM**), Faddeev-Leverrier's algorithm (**F-L**), the internal function of *Mathematica* (**Int**) and Danilevakii's method (**Dani**).

Table 1: CPU-Time (in second) for $d = 1$, $n = 3$ to 10

n	3	4	5	6	7	8	9	10
CHACM	0.04	0.08	0.21	0.40	0.74	1.16	1.97	3.12
F-L	0.08	0.28	0.63	1.31	2.68	4.51	7.24	11.32
Int	0.01	0.01	0.06	0.21	0.51	1.48	4.95	10.97
Dani	0.02	0.14	0.35	2.30	7.33	22.29	51.74	114.33

Table 2: CPU-Time (in second) for $d = 2$, $n = 3$ to 10

n	3	4	5	6	7	8	9	10
CHACM	0.06	0.17	0.34	0.56	1.02	1.78	2.93	4.62
F-L	0.23	0.53	0.12	1.54	2.77	5.11	9.14	23.96
Int	0.00	0.03	0.13	0.43	1.20	3.46	9.50	25.29
Dani	0.06	0.29	2.64	10.73	36.54	97.83	302.74	706.93

Table 3: CPU-Time (in second) for $d = 3$, $n = 3$ to 10

n	3	4	5	6	7	8	9	10
CHACM	0.07	0.13	0.31	0.55	1.08	1.69	2.95	4.73
F-L	0.17	0.57	1.42	2.72	5.37	9.26	15.04	24.32
Int	0.01	0.03	0.14	0.41	1.34	3.49	9.72	25.72
Dani	0.10	0.72	6.60	31.46	110.70	330.12	811.95	2308.33

Conclusion: A new Cayley-Hamilton theorem based method—CHACM has been presented. Theoretical analysis and preliminary computational tests show that it is efficient for computing characteristic polynomials of polynomial matrices.

Acknowlegement: Thank Professor S. Moritsugu for bringing the Danilevskii's method to our attention.

References

[1] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer-Verlag, Berlin, 1993.

[2] D.K. Faddeev and V.N. Faddeeva, *Computational Methods of Linear Algebra*, Freeman, San Francisco, 1963.

[3] G.H. Golub and C.F. Van Loan, *Matrix Computations*, John-Hopkins Univ., London, 1989.

[4] G. Helmberg, P. Wagner and G. Veltkamp, On Faddeev-Leverrier's Method for the computation of the characteristic polynomial of a matrix and of eigenvectors, *Linear Alg. Appl.*, **185**(1993), 219–233.

[5] T. Kitamoto, Efficient computation of the characteristic polynomial of a polynomial matrix, *Proc. of 27th Numerical Analysis Symposium of Japan*, 1997.

[6] K. Kuriyama and S. Moritsugu, Fraction-free method for computing rational normal forms of square matrices, *Trans. Japan Soc. Indust. Appl. Math.*, **4**(1996), 253–264.

[7] S. Moritsugu and K. Kuriyama, Fraction-free method for computing rational normal forms of polynomial matrices, *RISC Techreports*, 97–18 (available on internet by the address: ftp://ftp.risc.uni-linz.ac.at/pub/techreports/), 1997.

[8] V. Pan, Computing the determinant and the characteristic polynomial of a matrix via solving linear systems of equations, *Inform. Process. Lett.*, **28**(1988), 71–75.

[9] S. Rombouts and K. Heyde, An accurate and efficient algorithm for the characteristic polynomial of a general square matrix, *J. Comput. Phys.*, **140**(1998), 453-458.

[10] J. Wang and C. Chen, On the computation of the characteristic polynomial of a matrix, *IEEE Trans. Automat. Control*, **27**(1982), 449–451.

[11] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon, Oxford, 1965.

[12] D.-Z. Zheng, A new method on computation of the characteristic polynomial for a class of square matrices, *IEEE Trans. Automat. Control*, **28**(1983), 516–518.