Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

A Fast Algorithm for the Division of Two Polynomial Matrices

B.Codenotti , G.Lotti

Nota Interna B4-40

Ottobre 1988

A Fast Algorithm for the Division of two Polynomial Matrices

B.CODENOTTI(*) and G.LOTTI(**)

ABSTRACT

This paper presents a new algorithm for the division of two polynomial matrices, consisting of a modification of the algorithm shown by Wang and Zhou. We improve the performance of their algorithm by means of the use of convolutions, and therefore of FFT techniques. Our method is based on the fast inversion of block triangular Toeplitz matrices, and it is amenable for parallel implementation.

(*) Istituto di Elaborazione dell'Informazione
Via S.Maria, 46 - 56100-PISA (Italy).

(**)Dipartimento di Informatica
Corso Italia, 40 - 56100-PISA (Italy).

We consider the problem of performing the division of two matrix polynomials, as given in [4,5,7]. We follow the approach and the notation used in [6].

Indeed, let N(s) be an mxp polynomial matrix, and D(s) a pxp nonsingular polynomial matrix. Then there exist unique mxp polynomial matrices Q(s) and R(s) such that

$$N(s) = Q(s)\, D(s) + R(s), \qquad (1)$$

where $R(s)\,D^{-1}(s)$ is strictly proper.

Following [6], let D(s) be column reduced with column degrees u_i, i=1,2,...,p; let v_i, i=1,...,p, be the column degrees of N(s), and $u = \max\{u_i, i=1,\ldots,p\}$, $\ell = u + \max\{v_i - u_i, i=1,\ldots,p\}$.

It turns out that D(s), N(s), and R(s) can be written as:

$$D(s) = [\, D_u + D_{u-1}s^{-1} + \ldots + D_0 s^{-u} \,]\, H(s), \qquad (2)$$

$$N(s) = [\, N_\ell s^{\ell-u} + \ldots + N_0 s^{-u} \,]\, H(s), \qquad (3)$$

$$R(s) = [\, R_{u-\ell} s^{-\ell} + \ldots + R_0 s^{-u} \,]\, H(s), \qquad (4)$$

where D_u is nonsingular, $H(s) = \mathrm{Diag}\,(s^{w_1}, s^{w_2}, \ldots, s^{w_p})$.

Finally, let $r = \ell - u$, and

$$Q(s) = Q_r s^r + Q_{r-1}s^{r-1} + \ldots + Q_1 s + Q_0 . \qquad (5)$$

Substituting (2)-(5) into (1), we obtain the linear system:

$$A^T Q = N, \text{ where}$$

A is an (r+1)x(r+1) block Toeplitz upper triangular matrix, with square blocks of size p, namely

$$A = \begin{pmatrix} D_u & D_{u-1} & \cdots & D_{u-r} \\ 0 & D_u & D_{u-1} & \cdots & D_{u-r+1} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & \cdots & 0 & D_u & D_{u-1} \\ 0 & \cdots & 0 & 0 & D_u \end{pmatrix},$$

$$Q^T = [\ Q_r\ Q_{r-1}\ \ldots\ Q_0\ ],$$

and

$$N^T = [\ N_\ell\ N_{\ell-1}\ \ldots\ N_0\ ].$$

The first step of the algorithm we present consists of the inversion of matrix A^T. This matrix is nonsingular, since its diagonal block is nonsingular (see [6]). In general, it is not efficient to solve a linear system via the inversion of the coefficient matrix; on the contrary, in the special case under consideration, the properties of triangular Toeplitz matrices -to be analyzed next- make this approach computationally effective (as the cost-analysis to be performed) will put into evidence).

The inversion of matrix A^T can be carried out as follows:

1. Partition A^T as $\begin{pmatrix} A^T_{(r+1)p/2} & 0 \\ T & A^T_{(r+1)p/2} \end{pmatrix}$,
   and consider the linear system $A^T X = E$, where
   $X = [\ X_1^T\ X_2^T\ ]^T$, $E^T = [\ E_1^T\ 0\ ]$, with $E_1 = [\ I\ 0\ \ldots\ 0\ ]^T$.
   (The quantity $(r+1)p/2$ has been assumed to be integer; the general case can be handled with minor modifications).
2. Compute the matrix $X_1 = (A^T_{(r+1)p/2})^{-1}\ E_1$ (recursive step).
3. Compute the matrix $X_2 = -\ (A^T_{(r+1)p/2})^{-1}\ T\ X_1$.

Given that the inverse of a block triangular Toeplitz matrix is itself a block triangular Toeplitz matrix (see for example [1]),

it turns out that matrix X_1 contains all the information needed to represent matrix $(A^T_{(r+1),p/2})^{-1}$.

Other work has been performed in the field of triangular Toeplitz matrix inversion. The parallel algorithm suggested in [1] attains optimal time-cost, but it uses approximation, and it is much more intrigued. For the description of other algorithms dealing with fast inversion of certain Toeplitz matrices, see [1,2] and references therein.

Before evaluating the cost of the algorithm either in a sequential or in a parallel environment, we now present a convenient way to carry out step 3 of the algorithm.

Step 3 can be rewritten as:

$$X_2 = L_1 L_2 L_1 E_1 + L_1 U_2 L_1 E_1,$$

where $L_1 = -(A^T_{(r+1),p/2})^{-1}$, and $T = L_2 + U_2$, and

L_2 (U_2) is a block lower (upper) triangular Toeplitz matrix.

It is easy to see that a block lower (upper) triangular Toeplitz matrix can be row and column permuted, in order to obtain a (dense) block matrix whose blocks are lower (upper) triangular Toeplitz matrices.

Indeed, consider for example the product $r = Lx$, where

$x^T = [\, x_1^T \ldots x_k^T \,]^T$, with x_i p-vector, $i=1,2,\ldots,k$, and L is the kxk block lower triangular Toeplitx matrix

$$L = \begin{pmatrix} L_1 & 0 & \ldots & 0 \\ L_2 & L_1 & 0 & \ldots 0 \\ \ldots & \ldots & \ldots & \ldots \\ L_{k-1} & \ldots & & 0 \\ L_k & \ldots & & L_1 \end{pmatrix}, \quad \text{with } L_i \text{ pxp matrix, } i=1,\ldots,k.$$

The computation of Lx is equivalent, up to permutations, to the computation of $\tilde{r}=\tilde{L}\tilde{x}$, where $\tilde{r}=Pr$, $\tilde{L}=PLP^T$, $\tilde{x}=Px$, and P is an opportune $(kp)\times(kp)$ permutation matrix. More precisely, denoted by x_i the p-vector $x_i = [x_{i,1} \ \dots \ x_{i,p}]^T$, $i=1,\dots,k$, then $\tilde{x}=Px$ is the vector $\tilde{x} = [\,\tilde{x}_1^T \ \dots \ \tilde{x}_p^T\,]^T$, where $\tilde{x}_\ell$ is the k-vector

$\tilde{x}_\ell = [x_{1,\ell} \ \dots \ x_{k,\ell}]^T$, $\ell=1,\dots,p$.

Matrix $\tilde{L} = PLP^T$ is a dense $p\times p$ block matrix, whose blocks are $k\times k$ lower triangular Toeplitz matrices. Since the product of a triangular Toeplitz matrix by a vector is equivalent to a discrete open convolution, then it can be computed by means of FFT algorithms (see [1,2,3]).

The computational efficiency of the algorithm we propose resides in the fact that the convolution of a k-vector has sequential cost (measured as number of operations) $O(k\log k)$, and parallel cost $O(\log k)$ on k processors [1].

Our goal was the computation of the quotient and of the remainder of the division, and can be attained, after the computation of $(A^T)^{-1}$, according to the following stages.

1. Compute Q as the product $(A^T)^{-1} N$;
2. Compute $R = M - G^T \tilde{Q}$, where

$R^T = [\,R_0 \ \dots \ R_{d-1}\,]$, and

$$G = \begin{pmatrix} D_0 & D_1 & \dots & D_{d-1} \\ 0 & D_0 & D_1 & \dots & D_{d-2} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & 0 & D_0 & D_1 \\ 0 & \dots & 0 & 0 & D_0 \end{pmatrix},$$

$$M^T = [\ N_0\ N_1\ \ .\ .\ .\ \ N_{u-1}\ ],$$

and

$$\bar{Q}^T = [\ Q_0\ .\ .\ .\ Q_r\ \underbrace{0\ .\ .\ .\ 0}_{u-r-1}\ ],$$

by performing a product and an addition between rectangular matrices.

We now can present the analysis of the complexity of our algorithm, either in a sequential or in a parallel environment.

For what concerns the inversion of matrix A^T, the sequential cost in terms of arithmetic operations can be analyzed in terms of the recurrence relation:

$$C(rp) = C(rp/2) + O(p^\omega r \log r),$$

i.e $C(rp) = O(p^\omega r \log r)$.

The computation of matrix Q can be carried out with $O(p^\omega m r \log r)$ arithmetic operations, and that of R with $O(p^\omega m u \log u)$ operations, by performing mp^2 convolutions of order r, and u, respectively. The overall number of operations results thus to be of order

$$p^2 \max(m,p)\ (r \log r + u \log u).$$

A similar analysis shows that the parallel cost of our algorithm is given by

$$O(\log^2 rp)$$

time steps.

It is easy to show that the sequential cost of the algorithm presented in [6] is $p^{\omega} r \max(m,p) \max(r,u)$, and that such algorithm can not be significantly parallelized. Therefore our approach allows improving the computational performance of the algorithm in [6].

On the other hand, it is worth pointing out that the computational advantages of the recursive algorithm here described have their counterpart in the increased requirement in terms of amount of space (storage) needed. This disadvantage is rather obvious, since recursion naturally leads to larger memory requirements.

REFERENCES

[1] D.Bini, On the Parallel Solution of Certain Toeplitz Linear Systems, SIAM J.Comput. 13, pp.268-276 (1984).

[2] J.W.Cooley, P.A.W.Lewis, and P.D.Welch, Application of the Fast Fourier Transform to Computation of Fourier Integrals, Fourier series, and Convolution Integrals, IEEE Trans. Audio and Electroacoustics, AU-15, pp.79-84 (1967).

[3] A.K.Jain, Fast Inversion of Banded Toeplitz Matrices by Circular Decomposition, IEEE Trans. Acoust. Speech and Signal Process. 26(2), pp.121-126 (1978).

[4] I.S.Krishnarao and C.T.Chen, Two Polynomial Matrix Operations, IEEE Trans.Automat.Contr., Vol.AC-29, pp.346-348 (1984).

[5] S.Y.Zhang and C.T.Chen, An Algorithm for the Division of Two Polynomial Matrices, IEEE Trans.Automat.Contr., Vol.AC-28, pp.238-240 (1983).

[6] B.G.Wang and C.H. Zhou, An Efficient Division Algorithm for Polynomial Matrices, IEEE Trans.Automat.Contr., Vol.AC-31, pp.165-166 (1986).

[7] W.A.Wolovich, A Division Algorithm for Polynomial Matrices, IEEE Trans.Automat.Contr., Vol.AC-29, pp.656-658 (1984).