Integrated Intelligent Research (IIR)
International Journal of Computing Algorithm
Volume: 06 Issue: 01 June 2017 Page No.55-58
ISSN: 2278-2397

Matrix Power Computation Band Toeplitz Structure

Hamide Dogan, Luis R. Suarez
University of Texas at El Paso, Mathematics, El Paso, Texas, USA
Email: hdogan@utep.edu, lrsuarez@miners.utep.edu

Abstract - A simple algorithm for computing the larger positive integer powers m (>n) of an nxn matrix is discussed in this paper. Most recent algorithms are mentioned, and comparison data on the theoretical complexity of these algorithms, and the runtime data are provided.

Keywords: Matrix Power; Algorithms; Algorithm Complexity; Runtime; Band Toeplitz Matrices
AMS Classifications: 15-04; 15B05; 15A24; 15A03; 15A99

I. INTRODUCTION

Computing powers of square matrices is needed in many areas such as economics, statistics and bioinformatics. So far, the existing matrix power algorithms require computationally taxable approaches, many of which require eigenvalues. Thus, it can be tedious and expensive processes. Furthermore many others require distinct number of eigenvectors. For many matrices, this fails.An Iterative algorithm has been recently proposed by Abu-Saris and Ahmed [4]. This approach bypasses the eigenvalue computations. Hence, it also does not care whether a matrix has distinct eigenvectors. The particular algorithm makes use of an iterative approach in computing the coefficient of a remainder polynomial emerging from a division algorithm. Even though their approach does not require eigenvalue computations, due to its iterative nature, the process can be computationally taxing. The distinguishing feature of these algorithms however is that the computation of A^k (k>n) does not depend upon the computation of the powers (>n) of the nxn matrix A contrary to the existing methods that are integrating the classical matrix multiplication algorithms such as Winograd and Strassen algorithms [1-3]. In this paper, a simpler algorithm for computing any positive integer power k (> n) of nxn matrices is described based upon the ideas developed by Abu-Saris and Ahmed as well as others [4-5]. The characteristic of our algorithm is that not only the computation of A^k does not depend upon the computation of the powers (> n), it also does not require expensive iterative processes. We furthermore provide the comparison on the theoretical complexity of the algorithms between the three different approaches, and the runtime comparison of the iterative algorithm and the new algorithm (NewAlg). We provide the number of multiplications used in each algorithms as the measure of their complexity.

II. PRELIMINARIES

2.1 Matrix Multiplication Algorithms

Matrix multiplication algorithms are used in many matrix power computations. There are various approaches to matrix multiplication. One of which is the Winograd algorithm as well as the Naïve algorithm [1-3].

2.1.1 Naive Algorithm

The naive algorithm is solely based on the basic mathematical definition for the multiplication of two matrices. To compute each entry in the final n×n matrix, we need exactly n multiplications and *n - 1* additions. Since each of the n^2 entries in the first matrix is multiplied by exactly *n* entries from the second matrix, the total number of multiplications is $n \times n^2 = n^3$, and the total number of additions is $(n - 1) \cdot n^2 = n^3 - n^2$. Thus, we classify the naive algorithm as an $O(n^3)$ algorithm. Applying the Naive algorithm to the m^{th} power of an nxn mat $(m-1)(n^3)$.

2.1.2 Winograd Algorithm

Winograd algorithm trades multiplications for additions, much like Strassen's method [1-2]. However, it is asymptotically the same as the naive algorithm. Instead of multiplying individual numbers as in the naive algorithm, Winograd algorithm uses pairwise multiplication of the couples of entries and then subtracts the accumulated error. That is, for an nxn matrices, C=AB,

$$C_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - A_i - B_j \quad \text{with}$$

$$A_i = \sum_{k=1}^{n/2} a_{i,2k-1} a_{i,2k} \quad \text{and} \quad B_j = \sum_{k=1}^{n/2} b_{2k-1,j} b_{2k,j}$$

Since A_i and B_j are precomputed only once for each row and column, they require only n^2 multiplications [3]. The final summation does require $O(n^3)$ multiplications, but only half of those in the naive algorithm. Thus, the total number of multiplications is reduced to $\frac{1}{2} n^3 + n^2$. For the m^{th} power computation then Winograd algorithm requires at most total multiplications:

$$(m-1)\left((n^3/2) + n^2\right)$$

2.2 Iterative Algorithm

Cayley Hamilton theorem coupled with the Division Algorithm is an essential component of the ideas used in the iterative algorithm by Abu-Saris and Ahmed as well as Elaydi and Harris [4-5].

Cayley Hamilton Thm:
Let A ∈ M(n, n) with the characteristic polynomial

$$p(x) = \det(xI - A) = x^n + c_1 x^{n-1} + c_2 x^{n-2} + \cdot\cdot\cdot + c_n.$$

Then,

$$p(A) = A^n + c_1 A^{n-1} + c_2 A^{n-2} + \cdot\cdot\cdot + c_n I = 0.$$

Division Algorithm:
Let g(x), and p(x) be the real-valued polynomials of degrees m ≥ n respectively. Then,

$$g(x) = q(x)p(x) + r(x) \tag{1}$$

In the recursive algorithm, Abu-Saris and Ahmed uses specifically the polynomials, $g(x)=x^m$ and p(x) as the characteristic polynomial of an nxn matrix A applied to (1) along with the Cayley Hamilton theorem [4].

International Journal of Computing Algorithm
Volume: 06 Issue: 01 June 2017 Page No.55-58
ISSN: 2278-2397

Outline of the iterative algorithm:

Consider an nxn real-valued matrix A $(n \geq 2)$, and let

$p(x) = \det(xI - A) = x^n + \sum_{i=0}^{n-1} a_j x^j$.

Then,

$A^m = q_m(A)p(A) + r_m(A)$ for $m \geq n$,
Next, by the Cayley Hamilton Thm:

$$A^m = r_m(A) = \sum_{i=0}^{n-1} b_j(m)A^j, \qquad (2)$$

These ideas result in the following iterative algorithm for the evaluation of the coefficients *bj(m)* in (2):

$$b_j(m+1) = -a_j b_{n-1}(m) + b_{j-1}(m) \quad \left(j = 0, \dots, n-1;\ b_{-1}(m) = 0\right). \qquad (3)$$

The iteration in (3) uses one multiplication per coefficient, totaling n multiplications for a single power. Hence, for the mth power of an nxn matrix A, the number of multiplications carried out by (3) is:

$$(m-n).n+(n-2)(1/2n^3+n^2)+(n-1)n^2 \qquad (4)$$

Matrix polynomial in (2) requires the computation of the (n-1)th power of A. (3) provides the coefficients of the matrix polynomial for the powers <n. In (4), we consider the Winograd algorithm to calculate the number of multiplications used in matrix multiplications. Note that there is no need to repeat the matrix multiplication for each power in (2). Thus, only the number of multiplications used in the evaluation of the (n-1)th power of A. Then, the total number of multiplications used is: $(n-2)(1/2n^3+n^2)$. Finally, (2) uses at most $(n-1)\, n^2$ scalar matrix multiplications.

III. DEVELOPMENT OF THE NEW ALGORITHM (NEWALG)

Our algorithm expends the ideas from Abu-Saris and Ahmad [4]. Rather than applying the long division ideas resulting in the iteration (3), our algorithm applies vector space tools, in turn, solves a linear system for the coefficients of the remainder polynomial associated with the matrix power to be computed.

Given a characteristic polynomial, $F(x)=\sum_{i=0}^{n} y_i x^i$, of a nxn matrix A. Consider the real vector space of the polynomials of degree m (>n), and let's consider its basis $\{1, x, x^2, \dots, x^{n-1}, F, xF, x^2F, \dots, x^{m-n}F\}$. Then, for any polynomial of degree m, $G(x)=\sum_{i=0}^{m} z_i x^i$, we obtain :

$$G(x)=\sum_{i=0}^{m} z_i x^i = \sum_{i=0}^{n-1} a_i x^i + \sum_{i=0}^{m-n} b_i x^i F \qquad (5)$$

Note that (5) is equivalent to the division algorithm $G(x) = q(x)F(x) + r(x)$ where $r(x)=\sum_{i=0}^{n-1} a_i x^i$. The difference between the two ideas is that (5) does not require expensive iterative steps. In fact, it needs only basic matrix algebra approaches to solving the linear system:

$$\sum_{i=0}^{k} b_{m-n-k+i} y_{n-i} = z_{m-k} \quad \text{where } k = 0, \dots, m-n \qquad (6)$$

Thus, the system (6) is represented by an (m-n+1)x(m-n+2) coefficient matrix that has a structure very similar to that of the Toeplitz Matrices.

$$\begin{bmatrix} 0 & 0 & \cdots & 0 & y_n & z_m \\ 0 & \cdots & & y_n & y_{n-1} & z_{m-1} \\ 0 & & y_n & y_{n-1} & y_{n-2} & z_{m-2} \\ \cdots & \cdots & & \cdots & \cdots & \cdots \\ y_n & \cdots & & y_{n-(m-n)-1} & y_{n-(m-n)} & z_n \end{bmatrix} \qquad (7)$$

Solving (6) can be achieved via the basic back substitution process or via the row reduced echelon form of the matrix in (7). Then, one can use the solution (values of b_i) to (6) to obtain the coefficients of the polynomial $\sum_{i=0}^{n-1} a_i x^i$ in (5):

$$a_i = z_i - \sum_{k=0}^{i} b_{i-k} y_k, \quad i = 0, \dots, n-1. \qquad (8)$$

IV. COMPUTING A^m

Consider the polynomial $G(x) = x^m$ $(m > n)$. Again, let F(x) be the characteristic polynomial of an nxn matrix A. Then, (5) gives the matrix polynomial for the mth power (m>n) of A:

$$A^m = \sum_{i=0}^{n-1} a_i A^i \qquad (9)$$

In fact, the system (6) is represented by the matrix:

$$\begin{bmatrix} 0 & 0 & \cdots & 0 & 1 & 1 \\ 0 & \cdots & & 1 & y_{n-1} & 0 \\ 0 & & 1 & y_{n-1} & y_{n-2} & 0 \\ \cdots & \cdots & & \cdots & . & \cdots \\ 1 & \cdots & & y_{n-(m-n)-1} & y_{n-(m-n)} & 0 \end{bmatrix} \qquad (10)$$

Considering that the characteristic polynomial of A has at most n-nonzero coefficients, y_i, the matrix in (10) gets at most n-nonzero values in each row. This is due to its similarity to the band Toeplitz matrices.

EXAMPLE:

Let's consider $A=\begin{pmatrix} 0 & 1 & 1 \\ -2 & 3 & 1 \\ -3 & 1 & 4 \end{pmatrix}$. This matrix has two distinct eigenvectors. Its characteristic polynomial is $F(x)= x^3-7x^2+16x-12$. Applying (10) for the 8th power of A results in a 6x7 matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -7 & 0 \\ 0 & 0 & 0 & 1 & -7 & 16 & 0 \\ 0 & 0 & 1 & -7 & 16 & -12 & 0 \\ 0 & 1 & -7 & 16 & -12 & 0 & 0 \\ 1 & -7 & 16 & -12 & 0 & 0 & 0 \end{bmatrix} \qquad (11)$$

Solving (6), we obtain:

$b_0 = 1611$

$b_1 = 473$

$b_2 = 131$

$b_3 = 33$

$b_4 = 7$

$b_5 = 1$

Using (8), one can obtain the matrix polynomial whose evaluation results in A^8:

$A^8 = (19332)I-(20100)A+(5281)A^2 =$

$$\begin{bmatrix} -7073 & 1024 & 6305 \\ -7329 & 1280 & 6305 \\ -13634 & 1024 & 12866 \end{bmatrix} \quad (12)$$

Note that to compute A^8, our algorithm used about 55 multiplications as opposed to 189 multiplications with the Naive algorithm, about 157 multiplications with the Winograd algorithm.

Table 1. Multiplication Count (Matrix size n; Power m).

n	m	Naive	Wingrd	NewAlg
4.	10.	576.	432.	168.
4.	15.	896.	672.	188.
4.	20.	1216.	912.	208.
4.	25.	1536.	1152.	228.
4.	30.	1856.	1392.	248.
4.	35.	2176.	1632.	268.
4.	40.	2496.	1872.	288.
4.	45.	2816.	2112.	308.
4.	50.	3136.	2352.	328.

4.1 Theoretical Complexity of the algorithm for A^m

Looking at (11), one can see that the first and the second rows are not using any multiplication in the back-substitution process. The third row uses one multiplication associated with the entry value, **-7**. The forth row uses two multiplications associated with the entry values **-7** and **16**. This is because the value of b_5 is 1 from the first row. The rows 5 and 6 each uses 3 multiplications. Then, generalizing this pattern to an nxn matrix A, the maximum numbers of multiplication applied in our algorithm to solve (6) is: (n)(n-1)/2 + (m-2n)(n).

Additionally, (8) requires at most n(n+1)/2 many multiplications. Finally, (9) needs at most $(n-2)(\frac{n^3}{2}+n^2)+(n-1)n^2$ multiplications. Here, we used the Winograd algorithm for the matrix powers (< n). Altogether, the total number of multiplication for the mth (m>2n) power of an nxn matrix is at most:

$$n(n+1)/2+ (n-1)(n)/2 + (n)(m-2n) + (n-2)(\frac{n^3}{2}+n^2) + (n-1)n^2$$

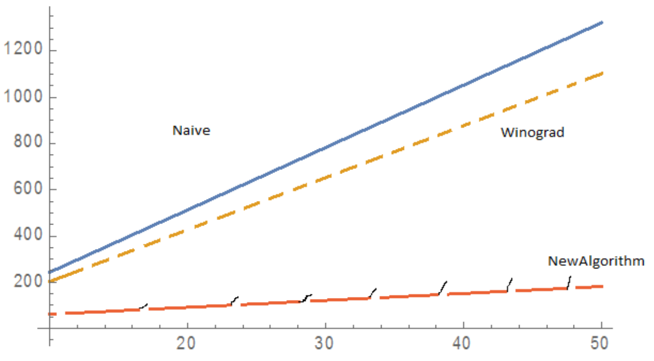

Figure 1. Comparison of the Numbers of Multiplication for 4x4 matrices.

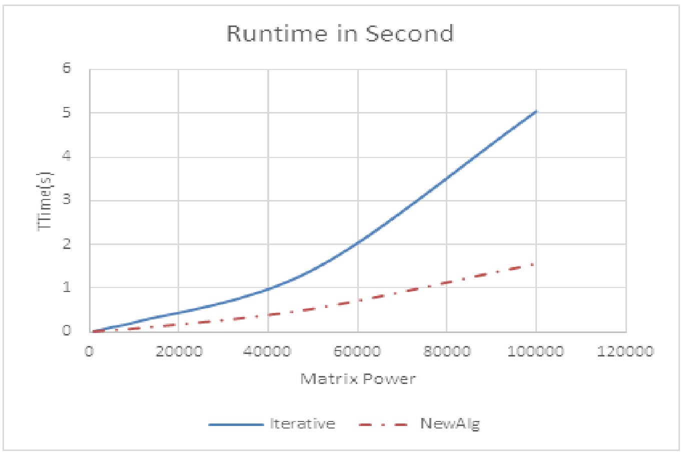

Figure 2. Runtime comparison for a 3x3 Matrix.

V. COMPARISONS

If we compare the number of multiplication applied by our algorithm (NewAlg.) against the Naive approach, and the Winograd algorithm [1-3], one sees in table 1 above that our algorithm outperforms both algorithms. That is, even though difference between the numbers of multiplication applied is small for the smaller powers of A, as the matrix power increases, the difference increases almost exponentially. See fig. 1 above for the multiplication count for 3x3 matrices and for the matrix powers ranging from 10 to 50.

As for the comparison of the NewAlg and the Iterative Algorithm, even though the Iterative algorithm and the NewAlgorithm use about the same number of multiplication, looking at the runtime values in seconds, one can see that the NewAlg performs significantly better especially for the higher matrix powers. This behavior is clearly depicted in Fig. 2 above. In fact, one can see that for the powers less than 2000, the two algorithms run at about the same speed with very little difference in the seconds it took for the algorithms to calculate the coefficients of the remainder polynomial in (2) and (9). After this point however it takes drastically more seconds for the Iterative algorithm to carry on calculations. We should note that the data displayed in fig. 2 was gathered using a Latitude E7440, Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz 2.40GHz in a *Mathematica* environment. See the Appendix for the *Mathematica* programs used in each of the two algorithms.

VI. CONCLUSION

NewAlg in comparison to the Naive and the Winograd algorithm uses significantly less numbers of multiplication [1-3]. Furthermore, in comparing to the Iterative process [4], the NewAlg performed in real-time significantly faster. Thus, integrating the primary ideas from NewAlg into the existing matrix power computation algorithms may prove to be an effective way to improve the runtime of the larger matrix powers.As a final remark, we should note that in our computations, we did not take into account the number of arithmetic operations used in the evaluation of the characteristics polynomials. We recognize that characteristic polynomials will add to the complexity of each of the algorithms. Since all the algorithms we discussed in the paper require the evaluation of the characteristic polynomial, this fact will not change our comparisons of the algorithm complexity.

References

[1] R. P. Brent (1970) Algorithms for matrix multiplications, Stan-CS (Tech. Rept. of the Computer Science Dept. of Stanford University, Stanford, CA), 1970, pp. 70-157.
[2] U. Manber (1989) Introduction to Algorithms: A Creative Approach. pp 301- 304. Pearson Education: New Jersey, 1989.
[3] S. Winograd (1968) A new algorithm for inner-product, IEEE Trans. Computers 17 (1968), 193-194.
[4] R. Abu-Saris and W. Ahmad (2005) Avoiding Eigenvalues in Computing Matrix Power, The American Mathematical Monthly , Vol. 112, Number 5, (May 2005) 450-454.
[5] S.N. Elaydi and W. A. Harris, Jr. (1997) On the computation of A^n, SIAM Rev. 40 (1998) 965-971.

Appendix

Mathematica Programs for the Iterative and the New Algorithm

Iterative Algorithm

```
In[3]:= k = 3; a[0] = -12; a[1] = 16; a[2] = -7;
        a[3] = 1; A[0, 0] := -a[0];
        A[1, 0] := -a[1];
        A[2, 0] := -a[2];
```

$$\text{Print}\left[\sum_{i=0}^{k} a[i]\, x^{i}\right];$$

```
        AbsoluteTiming[
         Do[
          Do[
           A[i, j] = If[i - 1 < 0,
             -A[k - 1, j - 1] a[i],
             -A[k - 1, j - 1] a[i] + A[i - 1, j - 1]],
           {i, 0, 2}], {j, 1, 100000, 1}]]
```

New Algorithm (NewAlg)

```
h = 100000;
a = {{0, 1, 1}, {-2, 3, 1}, {-3, 1, 4}};
- CharacteristicPolynomial[a, x];
p0 = 1;
p1 = Coefficient[-CharacteristicPolynomial[a, x], x^2]; p2 = Coefficient[-CharacteristicPolynomial[a, x], x];
p3 = -CharacteristicPolynomial[a, x] - p0 x^3 - p1 x^2 - p2 x; f[i_] := -p1 f[i - 1] - p2 f[i - 2] + p3 f[i - 3];
m = h; n = 3;
N[AbsoluteTiming[{f[0] = 1;
   f[1] = -p1 f[0];
   f[2] = -p1 f[1] - p2 f[0];
   f[3] = -p1 f[2] - p2 f[1] - p3 f[0];
   Do[f[i] = -p1 f[i - 1] - p2 f[i - 2] - p3 f[i - 3], {i, 4, m - n}]; c = f[m - n - 2];
    d = f[m - n - 1]; k = f[m - n]; (-c*p3 - d*p2 - k*p1); (-p3*d - p2*k); -(k*p3)}
 ]]
```