A NOTE ON THE EVALUATION OF MATRIX POLYNOMIALS

by

Charles Van Loan

TR78-353

Department of Computer Science
Cornell University
Ithaca, New York 14853

A NOTE ON THE EVALUATION OF MATRIX POLYNOMIALS

Charles Van Loan

Abstract:

The problem of evaluating a polynomial $p(x)$ in a matrix A arises in many applications, e.g. the Taylor approximation of e^A. The $O(\sqrt{q}\, n^3)$ algorithm of Paterson and Stockmeyer has the drawback that it requires $O(\sqrt{q}\, n^2)$ storage, where q is the degree of p and n is the dimension of A. An algorithm which greatly reduces this storage requirement without undue loss of speed is presented.

Author's Current Address:

Department of Computer Science
Cornell University
Ithaca, New York 14853

A NOTE ON THE EVALUATION OF MATRIX POLYNOMIALS

Charles Van Loan

One method for evaluating the matrix polynomial

$$p(A) = b_0 I + b_1 A + \ldots + b_q A^q \qquad b_i \in R \ , \ A \in R^{nxn}$$

is to use Horner's rule:

$$S_0 = b_q I$$

For $k = 1,2,\ldots,q$

$$\lfloor \ S_k = AS_{k-1} + b_{q-k} I$$

whereupon $S_q = p(A)$. Ignoring low order terms in n, this algorithm requires $2n^2$ storage and $(q-1)n^3$ multiplications.

In an interesting paper by Paterson and Stockmeyer [2] , it is shown how this computation can be rearranged so that only $O[\sqrt{q}\ n^3]$ multiplications are needed. The main idea behind their algorithm is to apply Horner's rule to $p(A)$ regarded as a polynomial in A^s where $s \approx \sqrt{q}$. For example, if $q = 9$, then

$$p(A) = (b_9 I)(A^3)^3 + (b_8 A^2 + b_7 A + b_6 I)(A^3)^2 +$$

$$(b_5 A^2 + b_4 A + b_3 I)(A^3) + (b_2 A^2 + b_1 A + b_0 I)$$

Once A^2 and A^3 are computed, then only $2n^3$ multiplications are

required to evaluate $p(A)$ as evidenced by the following "Horner regrouping" :

$$p(A) = A^3[\, A^3[\, b_9A^3 + (b_8A^2 + b_7A + b_6I)] + (b_5A^2 + b_4A + b_3I)] + (b_2A^2 + b_1A + b_0I) .$$

Hence, by this technique, only $4n^3$ multiplications are required to evaluate $p(A)$. This is in contrast to the count of $8n^3$ associated with the "ordinary" application of Horner's rule.

In general, if s is any integer satisfying

$$1 \leq s \leq q$$

and

$$r = \left[q/s \right] = \text{the integral part of } q/s$$

then

$$p(A) = \sum_{k=0}^{r} B_k \, (A^s)^k \tag{1}$$

where

$$B_k = b_{sk+s-1} A^{s-1} + \ldots + b_{sk+1} A + b_{sk} I \qquad (k = 0,\ldots,r-1)$$

and

$$B_r = b_q A^{q-sr} + \ldots + b_{sr+1} A + b_{sr} I$$

Applying Horner's rule to (1) we obtain:

$$F_0 = B_r$$

For $k = 1,\ldots,r$

$$\lfloor F_k = (A^s)F_{k-1} + B_{r-k}$$

where $F_r = p(A)$. The fact that there can be a significant reduction in work as a result of these manipulations is made clear when this iteration is spelled out in greater detail:

Algorithm I

1. $Y_k = A^k \quad (k = 0,1,\ldots,s)$

2. $F_0 = b_q Y_{q-sr} + b_{q-1} Y_{q-sr-1} + \ldots + b_{sr+1} Y_1 + b_{sr} Y_0$

3. For $k = 1,\ldots,r$

$$\lfloor F_k = Y_s F_{k-1} + b_{s(r-k)+s-1} Y_{s-1} + \ldots + b_{s(r-k)} Y_0$$

where $F_r = p(A)$.

This is essentially Algorithm "B" in [2] applied to the evaluation of matrix polynomials. Noting that $(s-1)n^3$ multiplications are required to compute the Y_k and that there is one matrix-matrix multiplication per iteration in step 3, we conclude that for a given s, Algorithm I requires $w_1(s)n^3$ multiplications where

$$w_1(s) = \begin{cases} s + \left[q/s\right] - 2 & \text{if } s \text{ divides } q \\ s + \left[q/s\right] - 1 & \text{otherwise} \end{cases}$$

Setting $s = 1$ in Algorithm I renders the ordinary version of Horner's rule. Setting $s = [\sqrt{q}\,]$ approximately minimizes $w_1(s)$ and we then find $w_1([\sqrt{q}\,]) \approx 2\sqrt{q}$.

A discouraging feature of Algorithm I is that it requires $(s+1)n^3$ storage. (Arrays are required for $A, A^2, \ldots, A^s$, and the "current" F_k .) However, a slightly less efficient algorithm which requires only $3n^2$ storage results if $p(A)$ is computed one column at a time. To derive this algorithm, let e_j denote the j-th column of the identity matrix. From (1) we have

$$p(A)e_j = \sum_{k=0}^{r} (A^s)^k (B_k e_r)$$

which may be computed as follows:

$$f_0^{(j)} = B_r e_j$$

For $k = 1, \ldots, r$

$$\lfloor\ f_k^{(j)} = (A^s) f_{k-1}^{(j)} + B_{-k} e_j$$

where $f_r^{(j)} = p(A)e_j$, the j-th column of $p(A)$. Notice that once the vectors Ae_j , A^2e_j , ... , $A^{s-1}e_j$ are computed, then the vectors B_ke_j , $k = 0,...,r$ are "free" : .

$$B_ke_j = b_{sk+s-1}(A^{s-1}e_j) + ... + b_{sk+1}(Ae_j) + b_{sk}e_j \quad (k = 0,...$$

$$B_re_j = b_q A^{q-sr}e_j + ... + b_{sr+1}Ae_j + b_{sr}e_j$$

We therefore obtain the following algorithm:

Algorithm II

1. $Y = A^s$

2. For $j = 1,...,n$

   $y_0^{(j)} = e_j$

   For $k = 1,...,s-1$

      $y_k^{(j)} = A\, y_{k-1}^{(j)}$

   $f_0^{(j)} = b_q\, y_{q-sr}^{(j)} + ... + b_{sr+1}\, y_1^{(j)} + b_{sr}\, y_0^{(j)}$

   For $k = 1,...,r$

      $f_k^{(j)} = Y\, f_{k-1}^{(j)} + b_{s(r-k)+s-1}\, y_{s-1}^{(j)} + ... + b_{s(r-k)}y_0^{(}$

whereupon $p(A) = [\, f_r^{(1)} \mid f_r^{(2)} \mid ... \mid f_r^{(n)} \,]$.

Since arrays are needed for only A, A^s, and $p(A)$, Algorithm II requires $3n^2$ storage. A careful operation count for $s > 1$ shows that $w_2(s)n^3$ multiplications are required where

$$w_2(s) = \begin{cases} 2(s-1) + \left[q/s\right] - 2 & \text{if } s \text{ divides } q \\ 2(s-1) + \left[q/s\right] - 1 & \text{otherwise} \end{cases}$$

This work count is approximately minimized by setting $s = [\sqrt{q/2}]$ whereupon $w_2(s) \approx 2\sqrt{2q}$ - about 1.5 times the work associated with Algorithm I.

At the expense of one extra nxn array, A^s can be computed via its binary expansion

$$A^s = (A^{\beta_0 2^0})(A^{\beta_1 2^1})\cdots(A^{\beta_m 2^m}) \qquad s = \sum_{k=0}^{m} \beta_k 2^k$$

in $\alpha \log_2(s) n^3$ multiplications where $\alpha \in [1,2]$ depends upon the number of nonzero β_i. (If s is an exact power of two, the extra storage is not required.) When this feature is incorporated in Algorithm II, the work count becomes $\hat{w}_2(s)n^3$ where

$$\hat{w}_2(s) = \alpha \log_2(s) - 1 + w_1(s)$$

These ideas have been incorporated in a subroutine for approxmating the matrix exponential e^A with a truncated Taylor series

$$e^A \approx \sum_{k=0}^{q} A^k/k! = T_q(A)$$

For reasons based on the machine precision used (10^{-15}) , the norm of A , and the desired accuracy, q was chosen to be 16. (See [1] for an analysis of Taylor approximation to e^A .) Setting $s = 4$ and using binary squaring to compute A^4 , Algorithm II requires $7n^3$ multiplications and $3n^2$ storage to evaluate $T_{16}(A)$. This is less than half the work associated with traditional Horner evaluation and only marginally slower than the $6n^3$ multiplications needed by Algorithm I which requires $5n^2$ storage for this problem.

Acknowledgements

The author greatly profited from discussions he had with Dr. Len Berman who pointed out reference [2].

References

[1] C.B.Moler and C.Van Loan, Nineteen Dubious Way to Compute the Matrix Exponential, to appear, SIAM Review.

[2] M.S.Paterson and L.Stockmeyer, On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials, SIAM J.Computing, 2(1973),60-66.