Computers and Mathematics with Applications 59 (2010) 1532–1540

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Recursive formulation of the matrix Padé approximation in packed storage

M. Kaliyappan [a,*], S. Ponnusamy [b], S. Sundar [c]

[a] Maha College of Engineering, Salem-636 106, India
[b] Sona College of Technology, Salem 636 005, India
[c] Indian Institute of Technology, Madras-600 036, India

ARTICLE INFO

Article history:
Received 27 April 2009
Received in revised form 19 November 2009
Accepted 20 November 2009

Keywords:
Matrix Padé approximants
Recursive packed storage

ABSTRACT

The Extended Euclidean algorithm for matrix Padé approximants is applied to compute matrix Padé approximants when the coefficient matrices of the input matrix polynomial are triangular. The procedure given by Bjarne S. Anderson et al. for packing a triangular matrix in recursive packed storage is applied to pack a sequence of lower triangular matrices of a matrix polynomial in recursive packed storage. This recursive packed storage for a matrix polynomial is applied to compute matrix Padé approximants of the matrix polynomial using the Matrix Padé Extended Euclidean algorithm in packed form. The CPU time and memory comparison, in computing the matrix Padé approximants of a matrix polynomial, between the packed case and the non-packed case are described in detail.

1. Introduction

The Padé approximants, a class of rational functions, occupy a very prominent place in mathematics as a whole, mainly because of their interesting properties as well as their intrinsic connections with many branches of mathematics like number theory, the theory of functions, differential equations, asymptotics, orthogonal polynomials etc. [1–3]. Their fast convergence behaviour makes it possible to use them as a tool for predicting a function when only a limited number of coefficients of its power series expansions are known. Work on Padé or Padé-type approximants involves the explicit determination of the polynomials forming the numerator and denominator of the rational functions. There exist in the literature many methods for constructing the polynomials of the Padé approximants from the coefficients of the given power series [1–5]. Many physical problems which are concerned with perturbation expansions involving matrix coefficients, the problem of finding coefficients and methods for computing matrix Padé approximants from a formal power series with matrix coefficients are well studied [6–8]. Hence it is quite natural for us to apply these techniques for computing matrix Padé approximants in a case where the matrices are of very large order lower triangular type.

In Section 2 we have given the definition of matrix Padé approximants of the formal power series of matrix coefficients. Section 3 deals with the procedure for a lower triangular matrix in recursive packed form and packing of a sequence of lower triangular matrices in recursive packed form. An algorithm for finding the recursive inverse of a lower triangular matrix in recursive packed form is given in Section 4. The multiplication of two lower triangular matrices in recursive packed form is presented in Section 5. Section 6 is concerned with the algorithm for computing matrix Padé approximants in packed form. Section 7 focuses on CPU time and memory comparison in computing the matrix Padé approximants. The subroutines used in C language are presented in Section 8. The system configuration is given in the Appendix. The work presented in this paper is an extension of that presented in [6]. Throughout this paper the following symbols are used:

* Corresponding author. Tel.: +91 0427 2263804.
E-mail addresses: kaliprem@yahoo.co.in (M. Kaliyappan), ponsam@yahoo.com (S. Ponnusamy), slnt@iitm.ac.in (S. Sundar).

0898-1221/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2009.11.015

$[f(x)] \rightarrow f(x)$ is the polynomial in which the coefficients are square matrices of the same order n;
$I_n \rightarrow$ the identity matrix of order n;
$\emptyset_n \rightarrow$ the null matrix of order n;
$[RP\_S(x)] \rightarrow S(x)$ is the polynomial in which the coefficients are triangular matrices of the same order n and each coefficient matrix is recursively packed and also all the recursively packed matrices are packed according to their degree.

2. The matrix Padé approximants

The $[M/N]$ matrix Padé approximant of a formal power series

$$[S(x)] = \sum_{i=0}^{\infty} (s_n)_i \, x^i \tag{1}$$

where the $(s_n)_i$ are coefficients of the power series which are square matrices of the same order n is defined as the rational function $[P_M(x)]/[Q_N(x)]$ such that

$$[S(x)] = [P_M(x)]/[Q_N(x)] + O\left(x^M + N + 1\right). \tag{2}$$

The numerator and denominator of this matrix Padé approximant are

$$[P_M(x)] = \sum_{i=0}^{M} (p_n)_i \, x^i \tag{3}$$

and

$$[Q_N(x)] = \sum_{i=0}^{N} (q_n)_i \, x^i, \qquad [Q_N(0)] = I_n \tag{4}$$

where $[P_M(x)]$ and $[Q_N(x)]$ are polynomials of degree at most M and N respectively. If the degrees of $[P_M(x)]$ and $[Q_N(x)]$ are exactly M and N then $[P_M(x)]/[Q_N(x)]$ is called the normal matrix Padé approximant of order (M, N) and it is symbolically denoted by $(M/N)_{[S]}(x)$. We may define two kinds of Padé approximants, since the $(s_n)_i$ need no longer commute, through the equations

$$[S(x)] - [P_M(x)]\,[Q_N(x)]^{-1} = O\left(x^{M+N+1}\right) \tag{5}$$

or

$$[S(x)] - [Q_N(x)]^{-1}\,[P_M(x)] = O\left(x^{M+N+1}\right) \text{ (by Eq. (2))}. \tag{6}$$

However we can show that these two Padé approximants are actually identical. For different chosen values of $M(\geq 0)$ and $N(\geq 0)$ we can construct the Padé table in which the distinct approximants of the function $[S(x)]$ would be the elements.

There arise mainly two problems. One is the *coefficient problem*, in which it is required to find the numerator and denominator coefficient matrices, $(p_n)_i$ and $(q_n)_i$, of Eqs. (3) and (4) for the input $(s_n)_i$ of Eq. (1). The other is the *value problem*, which is that of determining the value of the desired approximant for an explicit input chosen for x. In this paper we mainly concentrate on the first problem for a special case where the matrix coefficients involved are lower triangular.

3. The lower triangular matrix in recursive packed storage

A symmetric or triangular matrix may be stored in recursive packed form. The advantage of storing a triangular or symmetric matrix in recursive packed form was given in [9]. We present here the method for storing the lower triangular matrix in recursive packed form.

For the recursive packed form, we first store the triangular matrix in packed form and then convert to the recursive packed form. In order to store a lower triangular matrix in packed form, the columns of the triangular matrix are stored sequentially in a one-dimensional array starting with the first column. The mapping between positions in full storage and in packed storage for a lower triangular matrix of size n is given below:

$A_{i,j}$	i, j	UPLO
$AP_{i+(j-1)(2n-j)/2}$	$1 \leq j \leq n, j \leq i \leq n$	L
For UPLO $= L$, lower triangular.		

The advantage of this storage form is the saving of nearly half the memory as compared to full storage.

For converting from packed storage to recursive packed storage we do a reordering process. In this process the matrix is divided into two parts, namely a trapezoidal and a triangular part, by columns. The trapezoid is formed by the first p columns ($p = \lfloor n/2 \rfloor$) and the triangle by the remaining $n - p$ columns (Fig. 1). Keeping the triangle part in packed storage, the trapezoidal part is reordered. In the trapezoidal part, the triangle portion of it is in packed storage and the rectangle portion in full storage. The reordering demands a buffer of the size $p(p-1)/2$. The reordering has the following steps.

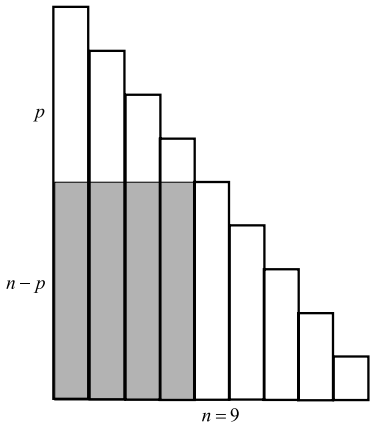

Fig. 1. A lower triangular matrix of order n ($n = 9$).

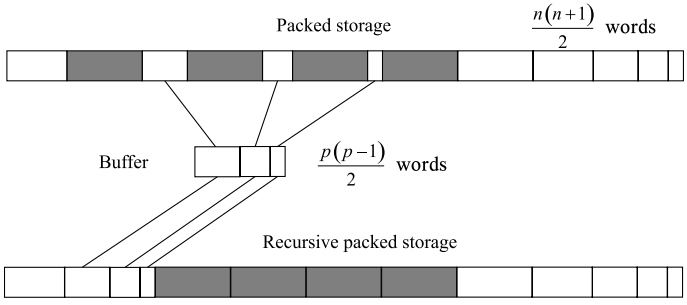

Fig. 2. Converting from packed storage to recursive packed storage.

First, the last $p - 1$ columns of the triangular part of the trapezoid are copied to the buffer and move the elements of the rectangular part of the trapezoid to the right from next to the last column; finally the buffer is copied back to the top of the trapezoid (shown in Fig. 2).

The addresses of the first elements in the leading triangular submatrix, the rectangular submatrix, and the trailing triangular submatrix are respectively given by

- 1,
- $1 + (p(p + 1)/2)$,
- $1 + np - (p(p - 1)/2)$.

After the reordering, the leading and trailing triangles are both in the same lower packed storage scheme as the original triangular matrix. The rectangular part of the reordered matrix is now kept in full matrix storage. If desired, this offers an excellent opportunity to transpose the matrix while it is transformed to the recursive packed format. If the rectangular submatrix is square the transposition can be done completely in place. If it deviates from a square by a column, a buffer of the size of the columns is necessary to do the transposition; for this purpose we can reuse the buffer used for the reordering. The method of reordering is applied recursively to the leading and trailing triangles which are still in packed storage, until finally the original triangular packed matrix is divided into rectangular submatrices of decreasing size, all in full storage.

3.1. Packing of the coefficient matrices of the matrix polynomial $[S(x)]$

As in Section 3, a triangular matrix of order n can be represented as a row matrix $(1 \times n(n + 1)/2)$ in recursive packed form. If the coefficient matrices of a matrix polynomial $[S(x)]$ are triangular then each coefficient of a matrix polynomial $[S(x)]$ is individually packed recursively, in a one by one manner, according to the degree. The recursive packed matrices thus obtained are sequentially packed according to the degree of $[S(x)]$, resulting in a rectangular matrix, as shown in Fig. 3. The number of rows of the resulting rectangular matrix is the number of coefficient matrices of $[S(x)]$ and the number of columns is $n(n + 1)/2$.

4. The recursive triangular inverse in recursive packed form

To find the inverse of a lower triangular matrix in blocked form, usually we split the matrix A into three blocks A_{11}, A_{21} and A_{22} as shown in Fig. 4. Let B be the inverse of A; then $AB = BA = I$. From the above identity we get three block equations:

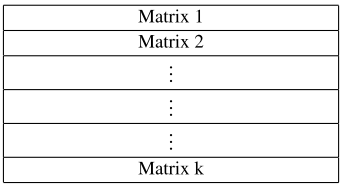

Fig. 3. Packing of k coefficient matrices of a matrix polynomial $[S(x)]$; each matrix is in recursive packed form.

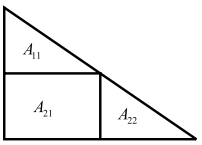

Fig. 4. Splitting of the lower triangular matrix A.

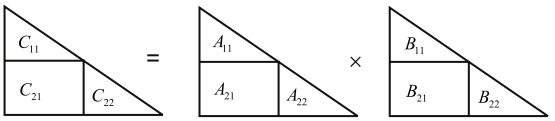

Fig. 5. Multiplication of two triangular matrices A and B in blocked form.

$A_{11}B_{11} = I, A_{21}B_{11} + A_{22}B_{21} = 0, A_{22}B_{22} = I$. The above three block equations imply that $B_{11} = A_{11}^{-1}, B_{22} = A_{22}^{-1}, B_{21} = -A_{22}^{-1}A_{21}A_{11}^{-1}$. Since the storage of the recursive packed form is as a leading triangular submatrix, a rectangular submatrix and a trailing triangular submatrix, in that order, the following algorithm finds the inverse of a lower triangular matrix recursively in recursive packed form.
Algorithm RP_RITM(A) (**R**ecursive **P**acked **R**ecursive **I**nverse of a **T**riangular **M**atrix)

INPUT: A is a lower triangular matrix of order n in recursive packed form.
OUTPUT: The inverse of A in recursive packed form.

```
1    if (n = 1) then
2        A = 1/A                          % since A is scalar
3    else
4        A11 ← RP_RITM (A11)              % invert A11
5        A22 ← RP_RITM (A22)              % invert A22
6        A21 ← A21 × A11                  % RP Triangular Rectangular Matrix Multiplication
7        A21 ← −A22 × A21                 % RP Triangular Rectangular Matrix Multiplication
8    end if
```

5. Recursive multiplication of two triangular matrices in recursive packed form

In order to multiply the two triangular matrices A and B in blocked form, usually we split A and B as shown in Fig. 5. Let $C = AB$, from which we obtain three block equations $C_{11} = A_{11}B_{11}, C_{21} = A_{21}B_{11} + A_{22}B_{21}$ and $C_{22} = A_{22}B_{22}$. Using the above three blocked equations we present an algorithm which multiplies A and B recursively in recursive packed form and returns C in recursive packed form.
Algorithm RP_TMTMM(A, B)(**R**ecursive **P**acked **T**riangular **M**atrix **T**riangular **M**atrix **M**ultiplication)

INPUT: Two triangular matrices A, B of order n in recursive packed form
OUTPUT: Triangular matrix C of order n in recursive packed form.

```
1    if (n = 1) then
2        C = A × B                        % since A and B are scalar
3    else
4        C11 ← RP_TMTMM (A11, B11)        % Recursive Packed Multiplication of A11 and B11
5        Ĉ21 ← A21 × B11                  % RP Triangular Rectangular Matrix Multiplication
```

6 $C_{21} \leftarrow \widehat{C}_{21} + (A_{22} \times B_{21})$ % RP Triangular Rectangular Matrix Multiplication and addition
7 $C_{22} \leftarrow$ RP_ TMTMM (A_{22}, B_{22}) % Recursive Packed multiplication of A_{22} and B_{22}
8 end if

6. Recursive formulation of the Matrix Padé Extended Euclidean algorithm in packed storage

The Extended Euclidean algorithm (EEA) is a well known algorithm, originally suggested by Aho et al. [10]. An application of the EEA to the ordinary non-matrix Padé case was presented by McEliece and Shearer [11] and Brent et al. [12]. The application of the Extended Euclidean algorithm to the matrix Padé case was presented by Achuthan and Sundar [6] for square matrix coefficients.

We present here a new algorithm **RFMAPEAP** (**R**ecursive **F**ormulation of **MA**trix **P**adé **E**xtended **E**uclidean **A**lgorithm in **P**acked storage) for computing matrix Padé approximants if the coefficient matrices of a matrix polynomial are triangular. The above algorithm is an extension of **MAPEA** (**MA**trix **P**adé **E**xtended **E**uclidean **A**lgorithm) presented in [6] by applying triangular matrix coefficients of the matrix polynomials in packed form as explained in Section 3.

Algorithm description

Name of the algorithm: **RFMAPEAP**(**R**ecursive **F**ormulation of **MA**trix **P**adé **E**xtended **E**uclidean **A**lgorithm in **P**acked storage)
INPUT: n, N, M, $[S(x)]$ where n is the order of the matrix, M is Maximum Padé numerator degree, N is the maximum Padé denominator degree, $[S(x)]$ is the series whose coefficients are triangular matrices.
OUTPUT: Sequence of anti-diagonal approximants starting from $(M + N, 0)$ to (M, N) if all the approximants exist.
Function **RFMAPEAP** ($[S(x)]$, degree_S, N, M, n)

```
1     degree _sum ← M + N
2     IF degree_S < degree _sum THEN exit 1
3     [a(x)] ← I_n x^(degree_sum+1)
4     [RP_S(x)]
5     [RP_a(x)]
6     [RP_b(x)] ← [RP_S(x)] mod [RP_a(x)]
7     IF N = 0 THEN exit 2
8     [dummy1(x)] ← Ø_n
9     RP_[dummy1(x)]
10    [dummy2(x)] ← I_n
11    RP_[dummy2(x)]
12    count ← 0
13    found ← false
      REPEAT
14      b_n ← Highest degree coefficient matrix of [ RP_b(x)]
15      IF (b_n^-1 not exists ) THEN exit 3
16        [RP_Q(x)] ←quotient ([ RP_a(x)], [ RP_b(x)])
17        [RP_R(x)] ← remainder ([ RP_a(x)], [ RP_b(x)])
18        IF ([RP_R(x)] = Ø_n) THEN ( found ← true)
          ELSE BEGIN
19          FOR index = 2 TO degree_[RP_Q(x)] DO
20            count ← count +1
21          IF count ≥ N THEN found ← true
            ELSE BEGIN
22              [RP_R1(x)] ← [RP_dummy1(x)] − [RP_Q(x)] × [RP_dummy2(x)]
23              count ← count +1
24              Padé _ approximant ← [RP_R(x)]/[RP_R1(x)]
25              OUTPUT [ Padé approximant (x)]
26              IF  count  <> N THEN
                BEGIN
27                [ RP_a(x)] ← [RP_b(x)]
28                [ RP_b(x)] ← [RP_R(x)]
29                [ RP_dummy 1(x)] ← [RP_ dummy 2(x)]
30                [RP_ dummy 2(x)] ← [RP_R1(x)]
                END
            END
          END
```

31 UNTIL(count $\geq N$) OR (found = true)
END (of RFMAPEAP)
exit 1: Insufficient degree for the input matrix polynomial $[S(x)]$,
Padé approximant cannot be found.
exit 2: As the Padé denominator degree is zero,
$[b(x)]$ itself is the required Padé approximant.
exit 3: As the inverse of b_n does not exist, Padé approximant cannot be found.

7. Calculation of the CPU time and memory comparison

7.1. CPU time

We have computed the matrix Padé approximants for the semi-normal power series

$$[S(x)] = I + Ix + Ix^2 + Ix^4 + Ix^8 + Ix^{16} + \cdots \tag{7}$$

used in [13] by using the algorithms **MAPEA** from [6] and **RFMAPEAP** for various matrix orders and various Padé orders when $x = 1$. We have considered identity matrices in (7) as triangular matrices while using the **RFMAPEAP** algorithm.

CPU times for computing (7/7) matrix Padé approximants for the semi-normal power series (7) at $x = 1$ for various orders of square matrices are presented in Table 1 (Fig. 6).

CPU times for computing (7/7) matrix Padé approximants for the semi-normal power series (7) at $x = 1$ for various orders of triangular matrices in packed form are presented in Table 2 (Fig. 7).

From Tables 1 and 2, it is clear that the **RFMAPEAP** algorithm is very fast compared to **MAPEA** if the input matrix polynomial coefficients are triangular (Fig. 8).

Table 1
CPU times for computing (7/7) matrix Padé approximants for the semi-normal power series (7) at $x = 1$ for various orders of square matrices.

Matrix order	100	200	300	400	500	600	700	800	900	1000
CPU time (s)	1	7	29	61	162	336	672	1048	1662	2645

Table 2
CPU times for computing (7/7) matrix Padé approximants for the semi-normal power series (7) at $x = 1$ for various orders of triangular matrices in packed form.

Matrix order	100	200	300	400	500	600	700	800	900	1000
CPU time (s)	1	2	12	34	41	84	133	267	436	871

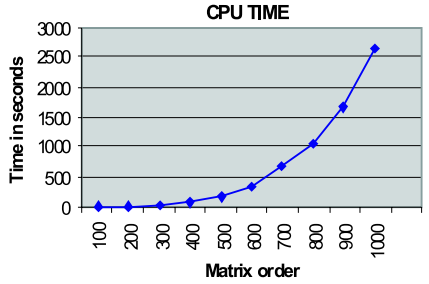

Fig. 6. Graphical representation of Table 1.

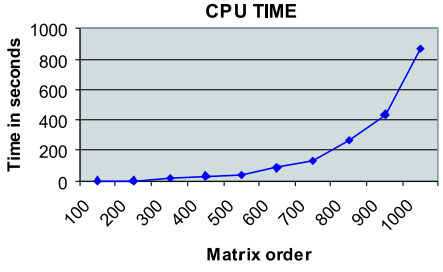

Fig. 7. Graphical representation of Table 2.

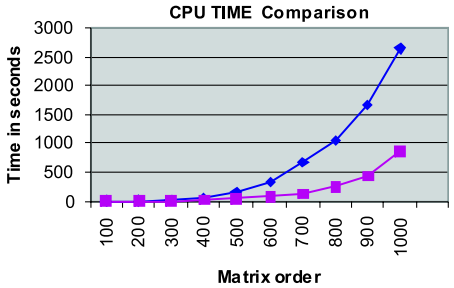

Fig. 8. Graphical representation for the comparison of Tables 1 and 2.

Table 3
Memory allocation for various matrix polynomials for computing (M/N) matrix Padé approximants.

S No	Name of the matrix polynomial	Memory needed to store the matrix polynomial
1	$[S(x)]$	$(M+N+2)n^2+M+N+2$
2	$[a(x)]$	$(M+N+2)n^2+M+N+2$
3	$[b(x)]$	$(M+N+2)n^2+M+N+2$
4	[dummy1(x)]	$(M+N+2)n^2+M+N+2$
5	[dummy2(x)]	$(M+N+2)n^2+M+N+2$
6	$Q(x)$	$(M+N+2)n^2+M+N+2$
7	$R(x)$	$(M+N+2)n^2+M+N+2$
8	$R1(x)$	$(M+N+2)n^2+M+N+2$

Table 4
Memory allocation for various subroutines for computing the (M/N) matrix Padé approximant.

S No	Name of the subroutine	Memory required to execute the subroutine
1	Matrix polynomial division	$6((M+N+2)n^2+M+N+2)+4n^2$
2	Matrix polynomial multiplication	$5((M+N+2)n^2+M+N+2)+2n^2$
3	Matrix polynomial addition	$((M+N+2)n^2+M+N+2)$
4	Matrix polynomial subtraction	$2((M+N+2)n^2+M+N+2)$
5	Matrix multiplication	n^2
6	Matrix inversion[a]	$6n^2$
7	Determinant[b]	n^2

[a] We have used the LU decomposition method to find the inverse of a matrix [14].
[b] We have used a modification of the Gauss elimination process for computing the value of the determinant of a matrix [15].

7.2. Memory comparison

The memory needed to store a matrix polynomial depends on its matrix coefficients and degree. Hence the memory needed to store a matrix polynomial is the product of the number of coefficients and order of the matrix and its degree. The differences in memory size for computing (M/N) matrix Padé approximants for the square matrix case and the triangular matrix case in packed form are presented below.

7.2.1. The memory required for computing the (M/N) matrix Padé approximant for the square matrix case

To get the (M/N) matrix Padé approximant, the degree of the input series $[S(x)]$ must be $M+N+1$; hence the number of coefficient matrices of $[S(x)]$ is at most $M+N+2$. So the memory needed to store $[S(x)]$ in the square matrix case is $(M+N+2)n^2+(M+N+2)$. Similarly, the memory allocations needed to store various matrix polynomials used for computing the (M/N) matrix Padé approximant are presented in Table 3.

We have used various subroutines for computing the (M/N) matrix Padé approximant. The memory allocations required to execute the subroutines are presented in Table 4.

The total memory that we require for computing the (M/N) matrix Padé approximant for the square matrix case is

$$22(M+N)n^2+58n^2+22(M+N+2).$$

7.2.2. The memory required for computing the (M/N) matrix Padé approximant for the triangular matrix case in packed storage

The memory required to store a triangular matrix of order n in recursive packed form is $n(n+1)/2$. To get the (M/N) matrix Padé approximants, the degree of the input series $[S(x)]$ must be $M+N+1$; hence the number of coefficient matrices of $[S(x)]$ is at most $M+N+2$. So the memory needed to store $[S(x)]$ in the packed form case is $(M+N+2)(n(n+1)/2)+M+N+2$. Similarly, the memory needed to store the various matrix polynomials used to compute the (M/N) matrix Padé approximant in packed form are presented in Table 5.

Table 5
Memory allocation for various matrix polynomials for computing the (M/N) matrix Padé approximant in packed form.

S No	Name of the matrix polynomial	Memory needed to store the matrix polynomial
1	[RP_ $S(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
2	[RP_ $a(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
3	[RP_ $b(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
4	[RP_ dummy1 (x)]	$(M+N+2)(n(n+1)/2)+M+N+2$
5	[RP_ dummy2 (x)]	$(M+N+2)(n(n+1)/2)+M+N+2$
6	[RP_ $Q(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
7	[RP_ $R(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
8	[RP_ $R1(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$

Table 6
Memory allocation for various subroutines for computing the (M/N) matrix Padé approximant in packed form.

S No	Name of the subroutine	Memory required to execute the subroutine
1	Packpoly_ division	$6(M+N+2)(n(n+1)/2)+6(M+N+2)+4(n(n+1)/2)$
2	Packpoly_ multiply	$5(M+N+2)(n(n+1)/2)+5(M+N+2)+2(n(n+1)/2)$
3	Packpoly_ add	$(M+N+2)(n(n+1)/2)+(M+N+2)$
4	Packpoly_ subtract	$2(M+N+2)(n(n+1)/2)+2(M+N+2)$
5	Recursive_ multiply	$(n(n+1)/2)$
6	Recursive_ inversion	$(n(n+1)/2)$
7	Recursive pack	$(n(n+1)/2)+p(p-1)/2$, where $p=\llcorner n/2\lrcorner$
8	Polynomial packing	$2(M+N+2)(n(n+1)/2)+2(n(n+1)/2)$

We have used various subroutines to compute the (M/N) matrix Padé approximants in packed form; the memory allocations required for executing the subroutines are presented in Table 6.

In addition we made a $4(M+N+2)n^2+4(M+N+2)$ storage space allocation for storing $[S(x)]$, $[a(x)]$, [dummy1(x)] and [dummy2(x)] before packing. Therefore the total memory that we require for computing the (M/N) matrix Padé approximant in packed storage is

$$24((M+N)(n^2+n)/2)+59(n^2+n)/2+4(M+N+2)n^2+26(M+N+2)+p(p-1)/2.$$

Hence the difference in memory size for computing the (M/N) matrix Padé approximants between the square matrix and triangular matrix cases is

$$6(M+N)n^2-12(M+N)n+(41/2)n^2-(59/2)n-4(M+N+2)-p(p-1)/2.$$

This quantity is positive starting from $M=0, N=1$ and $n=2$ or $M=1, N=0$ and $n=2$ up to the higher values of M, N and n. When n is large, for different values of M and N this quantity varies from nearly one quarter to nearly one third compared to the memory required for the square matrix case.

8. Subroutines used in C

The following subroutines are used to calculate the matrix Padé approximant in packed form:

1 Pack poly_ division: Returns the quotient and remainder of the two matrix polynomials in recursive packed form.
2 Pack poly_ multiply: Returns the result of multiplication of two matrix polynomials in recursive packed form.
3 Pack poly_ add: Returns the result of addition of two matrix polynomials in recursive packed form.
4 Pack poly_ subtract: Returns the result of subtraction of two matrix polynomials in recursive packed form.
5 Recursive_ multiply: Returns the result of multiplication of two triangular matrices in recursive packed form.
6 Recursive_ inversion: Returns the inverse of a triangular matrix in recursive packed form.
7 Recursive pack: Returns the recursive packed form of a triangular matrix.
8 Polynomial packing: Returns the packed form of the coefficient matrices of polynomial matrix.

Concluding remarks

We have shown here the methods used to construct matrix Padé approximants if the coefficient matrices of the input matrix polynomial are triangular, by using the recursive formulation of the Matrix Padé Extended Euclidean Algorithm in

Packed storage (**RFMAPEAP**). If the coefficients of the input matrix polynomial are triangular, the **RFMAPEAP** algorithm is very useful in getting matrix Padé approximants, instead of using the square matrix case algorithm **MAPEA**. This packed algorithm is very fast while using higher order triangular matrix coefficients of the matrix polynomial. Its memory usage is also considerably reduced compared to the square matrix case when the order of the matrix is large. This procedure can also be used for the upper triangular case.

Acknowledgement

The authors wish to thank the referee for his or her suggestions which improved the content of this article.

Appendix

The configuration of the system used to find the CPU time for computing the matrix Padé approximants for the semi-normal power series (7) is presented below.

System configuration:

Operating system:	Linux
Processor:	Intel (R) Pentium (R) 4 CPU 2.93 GHz
RAM:	256 MB
Cache size:	1 MB

References

[1] G.A. Baker Jr., Essentials of Padé Approximants, Academic Press, New York, 1975.
[2] G.A. Baker Jr., P.R. Graves-Morris, Padé Approximants, Part I: Basic Theory, in: Encyclopedia Math. Appl., vol. 13, Addison-Wesley, Reading, MA, 1981.
[3] W.B. Gragg, The Padé table and its relation to certain algorithms of numerical analysis, SIAM Rev. 14 (1972) 1–62.
[4] G. Claessens, A new look at the Padé table and the different methods for computing its elements, J. Comput. Appl. Math. 1 (1975) 141–152.
[5] L. Wuytack, Commented Bibliography on Techniques for Computing Padé Approximants, in: Lecture Notes in Mathematics, vol. 765, Springer, 1979, pp. 375–392.
[6] P. Achuthan, S. Sundar, A new application of the extended Euclidean algorithm for matrix Padé approximants, Comput. Math. Appl. 16 (4) (1988) 287–296.
[7] A. Bultheel, Recursive algorithm for matrix Padé problem, Math. Comp. 35 (1980) 875.
[8] J. Rissanen, Recursive evaluation of Padé approximants for matrix sequences, IBM J. Res. Dev. 401 (1972).
[9] Bjarne S. Anderson, Jerzy Wasniewski, Fred G. Gustavson, A recursive formulation of Choleskey factarization of a matrix in packed storage, ACM Trans. Math. Softw. 27 (2) (2001) 214–244.
[10] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass, 1974.
[11] Robert J. McEliece, James B. Shearer, A property of Euclid's algorithm and an application to Padé approximation, SIAM J. Appl. Math. 34 (4) (1978) 611–615.
[12] R.P. Brent, F.G. Gustavson, D.Y.Y. Yun, Fast solution of Toeplitz systems of equations and computation of Padé approximation, J. Algorithm. 1 (1980) 259.
[13] W. Leighton, W.T. Scott, A general continued fraction expansion, Bull. Amer. Math. Soc. 45 (1939) 596.
[14] M.K. Jain, S.R.K. Iyengar, R.K. Jain, Numerical Methods for Scientific and Engineering Computation, fourth ed., New Age International (P) Ltd, New Delhi, 2003.
[15] Robert J. Schilling, Sandra L. Harries, Applied Numerical Methods for Engineers Using MATLAB and C, second ed., Thomson Asia (P) Ltd., Singapore, 2002.