Journal of Mathematics Research
http: www.ccsenet.org/jmr
Vol. 1, No. 2
September 2009

Formulation of Matrix Pade Approximation in Rectangular Full Packed Storage

M. Kaliyappan (Corresponding author)
Maha college of engineering
Salem 636 106, India
E-mail: kaliprem@yahoo.co.in

S. Ponnusamy
Sona college of technology
Salem 636 005, India
E-mail: ponsam@yahoo.com

S. Sundar
Indian Institute of technology
Madras 600 036, India
E-mail: slnt@iitm.ac.in

Abstract

The Extended Euclidean algorithm for matrix Pade approximants is applied to compute matrix Pade approximants in rectangular full packed format (RFP) if the coefficient matrices of the input matrix polynomial are triangular. The procedure given by Gustavson et al for packing a triangular matrix in rectangular full packed format is applied to pack sequence of lower triangular matrices of a matrix polynomial in Rectangular Full Packed format. This RFP format of a matrix polynomial is applied to compute matrix Pade approximants of the matrix polynomial using Matrix Pade Extended Euclidean Algorithm. Algorithms for the multiplication of two triangular matrices and inverse of a triangular matrix in RFP format are also presented. The CPU time and memory comparison in computing the matrix Pade approximants of a matrix polynomial between RFP format case and non packed case are elucidated in detail.

Keywords: Matrix pade approximants, Rectangular Full Packed format

1. Introduction

Pade approximants have been the subject of much recent interest in many fields of applications (Baker, G.A. Jr., 1975). Some of the applications of this efficient technique is summation of infinite series, solution of integral equations, numerical inversion of Laplace transforms etc. Work on Pade or Pade-type approximants ultimately involves the explicit determination of the polynomials forming the numerator and denominator of rational functions can be found in (Gragg, W.B., 1972; Baker, G.A. Jr.,1975; Robert J. McEliece, James B. Shearer, 1978; Baker, G.A. Jr. and Graves - Morris, P.R.,1981). There exists in literature the use of extended Euclidean algorithm for matrix Pade approximants. Several researchers have worked on the problem of finding simple algorithm for computing matrix Pade approximants from a formal power series of matrix coefficients (Rissanen, J., 1972 ;Bultheel, A.,1980; Achuthan, P. and Sundar, S., 1988). So it is natural for us to apply these techniques for computing matrix Pade approximants in a special case where the matrices are of very large order lower triangular type. For the sake of completeness, the definition of matrix Pade approximants is presented in Section 2. Section 3 deals with the procedure for a lower triangular matrix in rectangular full packed format and packing of a sequence of lower triangular matrices in rectangular full packed format. Section 4 deals with the lower triangular inverse in rectangular full packed format. The multiplication of two lower triangular matrices in rectangular full packed format is presented in Section 5. Section 6 is concerned with the algorithm for computing matrix Pade approximants in rectangular full packed format. Section 7 brings out CPU time and memory comparison in computing the

matrix Pade approximants. The subroutines used in C language are presented in Section 8. The system configuration is given in appendix. The work presented in this paper is an advanced one that of presented in (Achuthan, P. and Sundar, S.,1988 , P. 287-296).

Throughout this paper the following symbols are used :
$[f(x)] \to f(x)$ is the polynomial in which the coefficients are square matrices of the same order n;
$I_n \to$ Identity matrix of order n;
$\emptyset_n \to$ Null matrix of order n;
$[RFP\_g(x)] \to g(x)$ is the polynomial in which the coefficients are triangular matrices of the same order n and each coefficient matrices are in rectangular full packed format and also all the rectangular full packed matrices are packed according to their degree.

2. The matrix Pade approximants

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

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

where $(s_n)_i$'s 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 Pade approximant are

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

and

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

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

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

or

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

However we can show that these two Pade approximants are actually identical.

For different chosen values of $M(\geq 0)$ and $N(\geq 0)$ we can construct the Pade 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 equations (3) and (4) for the input $(s_n)_i$ of equation (1); the other is the *value problem*, which is to determine the value of the desired approximant for an explicit input chosen for x. The algorithm to find matrix Pade approximants by using Extended Euclidean algorithm was given in (Achuthan, P. and Sundar, S.,1988 , P. 287-296).

3. Lower triangular matrix in rectangular full packed format

Symmetric or triangular matrix may be stored in rectangular full packed format. The advantage of storing symmetric or triangular matrix in Rectangular full packed format is explained in (Gustavson, G. and Jerzy Wasniewsky., 2007, P. 570 - 579). We presented here the method for storing a triangular matrix in Rectangular full packed format. Break the lower triangular matrix A into A_{11}, A_{21} and A_{22} as three sub matrices as shown in figure 1.

When n is even , the lower triangular sub matrix of A_{11} and the upper triangular sub matrix of A_{22}^T can be concatenated along their main diagonals into a $(k+1)\times k$ dense matrix. Then the square sub matrix A_{21} of order $k\times k$ is appended below the dense matrix $(k+1)\times k$, where $k = \lceil n/2 \rceil$. Thus the lower triangular matrix A can be stored as a $(n+1)\times k$ rectangular matrix. When n is odd, the lower triangular sub matrix A_{11} is concatenated with the upper triangular matrix A_{22}^T along their main diagonals into a $k\times k$ dense matrix. Then the rectangular sub matrix A_{21} of order $(k-1)\times k$ is appended below the $k\times k$ dense matrix. Thus A is stored as a $n\times k$ rectangular matrix. The above procedure is presented in figure 2.

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 stored as a $(n+1)\times k$ rectangular matrix if 'n' is even and $n\times k$ rectangular matrix if 'n' is odd in rectangular full packed format. If coefficient matrices of a matrix polynomial $[S(x)]$ are triangular then each coefficient matrices of $[S(x)]$ packed as Rectangular full packed format one by one according to their degree. Thus obtained Rectangular full packed matrices are sequentially packed according to the degree of the matrix polynomial $[S(x)]$ as shown in figure 3. Now the resulting matrix is also rectangular. If the number of coefficient matrices of $[S(x)]$ is m, the resulting rectangular matrix must be of order $(n+1)\times(mk)$, if n is even and $n\times(mk)$ if n is odd.

4. Triangular inverse in rectangular full packed format

There are many algorithms in literature to find the inverse of a lower triangular matrix. Let the inverse of lower triangular matrix A be D. Since A and D are in $2-$ by -2 blocking as shown in figure 4, from the identity $AD = DA = I$, we get three block equations $A_{11}D_{11} = I, A_{21}D_{11} + A_{22}D_{21} = 0, A_{22}D_{22} = I$. The above three block equations implies that $D_{11} = A_{11}^{-1}, D_{22} = A_{22}^{-1}, D_{21} = -D_{22}A_{21}D_{11}$. We find inverse of A_{11}, A_{22}^T using the algorithm given below separately and developed an algorithm to compute the inverse of A in rectangular full packed format. *Algorithm to find inverse of a lower triangular matrix in RFP form*

Name of the Algorithm: RFPTI (Rectangular Full Packed Triangular Inverse)
INPUT: Lower triangular matrix A in Rectangular full packed format.
OUT PUT: Inverse of lower triangular matrix A in Rectangular full packed format.

1	$D_{11} \leftarrow \texttt{In}(A_{11})$	% find inverse of A_{11}.
2	$D_{22}^T \leftarrow \texttt{In}(A_{22}^T)$	% find inverse of A_{22}^T (since A_{22} is A_{22}^T in RFP format)
3	$D_{21} \leftarrow \texttt{mul}(A_{21}, D_{11})$	% multiplication of A_{21} and D_{11}
4	$D_{21} \leftarrow (-1)\texttt{mul}(D_{22}, D_{21})$	% multiplication of D_{22} and D_{21}.

To find the inverse of matrix A_{11} of order k, we used the following algorithm in (Jeremy J Du Croz and Nicholas, Higham, J., 1992, P. 1- 19) which computes the inverse of A_{11}, column by column in the reverse order. Let the inverse of A_{11} be X.

Algorithm to find inverse of A_{11}
INPUT: Lower triangular matrix A_{11} of order k.
OUT PUT: Inverse of lower triangular matrix A_{11} of order k.

for $j = k : -1 : 1$
  $x_{jj} = \frac{1}{a_{jj}}$
  $X(j+1:k, j) = X(j+1:k, j+1:k)A_{11}(j+1:k, j)$
  $X(j+1:k, j) = -x_{jj}X(j+1:k, j)$
end

In order to find the inverse of A_{22}^T , the above algorithm modified to compute row by row in the reverse order.

5. Multiplications of two triangular matrixes in rectangular full packed format

As the lower triangular matrix A and B are in $2-$ by -2 block form as shown in figure 5, from $C = AB$, we obtain three block equations $C_{11} = A_{11}B_{11}, C_{21} = A_{21}B_{11} + A_{22}B_{21}, C_{22} = A_{22}B_{22}$.

Using the three block equations we presented an algorithm which multiplies A and B and returns the resulting matrix C in rectangular full packed format.

Name of the algorithm: RFPTRTRM (Rectangular Full Packed Triangular matrix and Triangular matrix Multiplication)
INPUT: Two lower triangular matrices A and B in Rectangular full packed format.
OUT PUT: Matrix C in Rectangular full packed format

1	$C_{11} \leftarrow$LTLTM(A_{11}, B_{11})	% multiplication of the lower triangular matrices A_{11} and B_{11}
2	$C_{22}^T \leftarrow$UTUTM$\left(B_{22}^T, A_{22}^T\right)$	% multiplication of the upper triangular matrices B_{22}^T and A_{22}^T (since A_{22}, B_{22} and C_{22} are A_{22}^T, B_{22}^T and C_{22}^T respectively in RFPF)
3	$\widehat{C}_{21} \leftarrow$mul$(A_{21}, B_{21})$	% multiplication of the matrices A_{21} and B_{11}
4	$C_{21} \leftarrow \widehat{C}_{21}+mul(A_{22}, B_{21})$	% multiplication of the matrices A_{22} and B_{21} and addition with $\widehat{C}_{21}$

Note 1: LTLTM = Lower Triangular matrix Lower Triangular matrix Multiplication.
UTUTM = Upper Triangular matrix Upper Triangular matrix Multiplication

6. Algorithm for computing matrix Pade approximants in Rectangular Full Packed format

The Extended Euclidean Algorithm (EEA) is a well known algorithm, originally suggested by Aho et al (Aho, A.V., Hopcroft, J.E., and Ullman, J.D., 1974). An application of EEA to ordinary non-matrix Pade was presented by McEliece and Sherarer (Robert J.McEliece, James B. Shearer., 1978, P. 611-615) and Brent et al (Brent, R.P., Gustavson,F.G., and Yun, D.Y.Y., 1980). The extended Euclidean Algorithm to matrix Pade was presented by P.Achuthan and S.Sundar in (Achuthan, P. and Sundar, S.,1988, P. 287-296) for square matrix coefficients.

We packed the triangular matrix coefficients of the matrix polynomials as explained in Section 3 and applied this packed

matrix polynomials in **MA**trix **P**ade **E**xtended **E**uclidean **A**lgorithm(**MAPEA**) in (Achuthan, P. and Sundar, S.,1988, P. 287-296).This algorithm gives a sequences of anti-diagonal approximants starting from $(M+N,0)$ to (M,N) in the Pade table if all the approximants exists. We presented here a new algorithm **MAPEARFP** (**MA**trix **P**ade **E**xtended Euclidean **A**lgorithm in **R**ectangular **F**ull **P**acked format) to compute matrix Pade approximants if coefficient matrices of a matrix polynomial are triangular.

Name of the algorithm: **MAPEARFP** (MAtrix Pade Extended Euclidean Algorithm in Rectangular Full Packed format)
INPUT: $n, N, M, [S(x)]$, n is order of the matrix, M is Pade numerator, N is Pade denominator, $[S(x)]$ is the Series whose coefficients are triangular matrices.
OUT PUT: Sequence of anti - diagonal approximants starting from $(M+N,0)$ to (M,N) if all the approximants exists.
Function MAPEARFP ([**S**(**x**)], **degree_ S**, **N**, **M**, **n**)

```
Degree _sum ← M + N
IF degree_S < Degree _sum THEN EXIT 1
[a(x)] ← I_n x^(degree_sum+1)
[RFP_S(x)]
[RFP_a(x)]
[RFP_b(x)] ← [RFP_S(x)] mod [RFP_a(x)]
IF N = 0 THEN EXIT 2
[dummy1(x)] ← ∅_n
[RFP_dummy1(x)]
[dummy2(x)] ← I_n
[RFP_dummy2(x)]
count ← 0
found ← false
REPEAT
    b_n ←Highest degree coefficient matrix of [ RFP_b(x)]
    IF (b_n^-1 not exists ) THEN EXIT 3
        [RFP_Q(x)] ←quotient ([ RFP_a(x)], [ RFP_b(x)])
        [RFP_R(x)] ← remainder ([ RFP_a(x)], [ RFP_b(x)])
        IF ([RFP_R(x)] = ∅_n) THEN ( found = true)
        ELSE BEGIN
          FOR index = 2 TO degree_[RFP_Q(x)] DO
              count ← count +1
          IF count ≥ N THEN found ← true
          ELSE BEGIN
                [RFP_R1(x)] ← [RFP_dummy1(x)] − [RFP_Q(x)] × [RFP_dummy2(x)]
                count ← count +1
                Pade _ approximant ← [RFP_R(x)]/[RFP_R1(x)]
                OUTPUT [ Pade approximant (x)]
                IF count < > N THEN
                BEGIN
                      [ RFP_a(x)] ← [RFP_b(x)]
                      [ RFP_b(x)] ← [RFP_R(x)]
                      [ RFP_dummy 1(x)] ← [RFP_ dummy 2(x)]
                      [RFP_ dummy 2(x)] ← [RFP_R1(x)]
                END
          END
        END
    UNTIL( count ≥ N) OR ( found = true)
    END (of MAPEARFP )
```

EXIT 1: In sufficient degree for the input matrix polynomial $[S(x)]$, so we cannot find approximants.
EXIT 2: As the Pade denominator degree is zero , $[b(x)]$ itself is the required Pade approximant.
EXIT 3: As the inverse of b_n does not exist, Pade approximant cannot be found.

7. Calculation of CPU time and memory comparison

7.1 CPU time

We have computed the matrix Pade approximants for the Semi - normal power Series

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

in (Leighton, W. and Scott,W.T., 1939) using the algorithms **MAPEA** in (Achuthan, P. and Sundar, S.,1988 , P.287-296) and **MAPEARFP** for various matrix orders and various Pade orders when $x = 1$. We have considered identity matrices in (7) as triangular matrices while using **MAPEARFP** algorithm.

CPU time to compute (7/7) matrix Pade approximants for the semi-normal power series (7) at $x = 1$ for various order of square matrices are presented in table 1 also the CPU time to compute (7/7) matrix Pade approximants for the semi-normal power series (7) at $x = 1$ for various order of triangular matrices in **RFP** format are presented in table 2.

While comparing the CPU time to compute matrix Pade approximants for the square matrix case using the algorithm **MAPEA** and for the triangular matrix case using the algorithm **MAPEARFP**, it is clear that **MAPEARFP** algorithm is very fast while using higher order triangular matrix coefficients of the matrix polynomial.

7.2 Memory comparison

The memory needed to store a matrix polynomial is depends on its matrix coefficients and degree. Hence the memory needed to store a matrix polynomial is (Number of coefficients matrices of the polynomial) (Memory for storing a coefficient matrix) + (Memory for storing exponents of the polynomial). The difference in memory size to compute (M / N) matrix Pade approximants for square matrix case and triangular matrix case in rectangular full packed storage are presented below.

7.2.1 The memory required to compute (M/N) matrix Pade approximants for square matrix case.

To get the (M/N) matrix Pade approximants, degree of the input series $[S(x)]$ must be $M + N + 1$, hence the number of coefficient matrices of $[S(x)]$ are at most $M + N + 2$. So the memory needed to store $[S(x)]$ in square matrix case is $(M + N + 2)n^2 + (M + N + 2)$. Similarly, the memory needed to store various matrix polynomials used to compute (M/N) matrix Pade approximants are presented in the table 3. In addition various subroutines are used to compute the matrix Pade approximants, the memory required to execute the subroutines are presented in table 4.

Total memory we required to compute (M/N) matrix Pade approximants for square matrix case is

$$\mathbf{24(M + N)n^2 + 56n^2 + 24(M + N + 2)}.$$

7.2.2 The memory required to compute the matrix Pade approximants for the triangular matrix case in rectangular full packed storage.

The memory required to store a triangular matrix of order 'n' in rectangular full packed format is $n(n + 1)/2$. To get the (M/N) matrix Pade approximants , the degree of the input the series $[S(x)]$ must be $M + N + 1$, hence the number of coefficient matrices of $[S(x)]$ are at most $M + N + 2$. So the memory needed to store $[S(x)]$ in Rectangular full packed format case is $(M+N+2)(n(n+1)/2)+M+N+2$. Similarly, the memory needed to store various matrix polynomials used to compute (M/N) matrix Pade approximants in rectangular full packed format are presented in the table 5. In addition we allocated $4(M+N+2)n^2+4(M+N+2)$ storage space to store $[S(x)]$, $[a(x)]$, $[dummy1(x)]$ and $[dummy2(x)]$ before packing. To compute matrix Pade approximants various subroutines are used, the memory required to execute the subroutines are presented in the following table 6.

The total memory we required to compute (M/N) matrix Pade approximants in rectangular full packed storage

$$\mathbf{25((M + N)(n^2 + n)/2) + 54(n^2 + n)/2 + 4(M + N + 2)n^2 + 28(M + N + 2)}.$$

Hence the difference in memory size to compute (M/N) Matrix Pade approximants between square matrix and triangular matrix cases is

$$\mathbf{(15/2)(M + N)n^2 - (25/2)(M + N)n + 21n^2 - 27n - 4(M + N + 2)}.$$

8. Subroutines used in C

The following subroutines were used to calculate matrix Pade approximants in rectangular full packed format:

1	Rect_ pack_Matrix _ multiply	:	Returns the result of multiplication of two triangular matrices in rectangular full packed format.
2	Rect_pack_Matrix _inversion	:	Returns the inverse of a triangular matrix in rectangular full packed format.
3	Rect_ pack	:	Returns the rectangular full packed format of a triangular matrix
4	Rect_ polynomial packing	:	Returns the packed form of the coefficient matrices of polynomial matrix
5	Rect_packpoly_ multiply	:	Returns the result of multiplication of two matrix polynomials in rectangular full packed format.
6	Rect_ packpoly_ addition	:	Returns the result of addition of two matrix polynomials in rectangular full packed format.
7	Rect_packpoly subtraction	:	Returns the result of subtraction of two matrix polynomials in rectangular full packed format.
8	Rect_packpoly_ division	:	Returns the quotient and remainder of the two matrix polynomials in rectangular full packed format.

CONCLUDING REMARKS

We have shown here how to construct normal matrix Pade approximants if the coefficients matrices of the input matrix polynomial are triangular matrix by using MAtrix Pade Extended Euclidean Algorithm in Rectangular Full Packed format **(MAPEARFP)**. If the coefficients of the input matrix polynomial are triangular, **MAPEARFP** algorithm is very useful in getting matrix Pade approximants instead of using 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 reduced nearly half compared to the square matrix case.

APPENDIX

The configuration of the system used to find the CPU time to compute matrix Pade 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

Achuthan, P. and Sundar, S. (1988). A new application of the extended Euclidean algorithm for matrix Pade approximants, *Comput. math. Applic*. 16(4). P. 287-296.

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (1974). *The Design and Analysis of Computer Algorithms*. Addition-Wesly, Reading, Mass.

Baker, G.A. Jr. (1975). *Essentials of Pade approximants*. New York: Academic press.

Baker, G.A. Jr. and Graves -Morris, P.R. (1981). Pade approximants, part I : Basic theory, *Encyclopedia Math*. Appl.13 Addition - Wesley, Reading, MA.

Bultheel, A. (1980). Recursive algorithm for matrix Pade problem. *Maths Comput*. 35, 875.

Brent, R.P., Gustavson,F.G., and Yun, D.Y.Y. (1980). Fast solution of Toeplitz systems of equations and computation of Pade approximations. *J.Algorithm* 1, 259.

Gragg, W.B. (1972). The Pade table and its relation to certain algorithms of numerical analysis. *SIAM Rev*. 14. P. 1-62.

Gustavson, G. and Jerzy Wasniewsky. (2007). Rectangular Full Packed Format for LAPACK Algorithms Timings on Several Computers. *Applied and parallel computing*, state the art in scientific computing, Springer. P. 570 - 579.

Jain, M.K., Iyengar, S.R.K., Jain, R.K. (2003). Numerical methods for Scientific and Engineering Computation. Fourth edition, New Delhi: *New Age International (P) Limited*.

Jeremy J Du Croz and Nicholas, Higham, J. (1992). Stability of methods for Matrix inversion. *IMA J. Numer. Anal.*, 12. P. 1- 19.

Leighton, W. and Scott, W.T. (1939). A general continued fraction expansion. *Bull.Amer. Math. Soc*, 45, 596.

Rissanen, J. (1972). Recursive evaluation of Pade approximants for matrix sequences. *IBM Jl Res. Dev*. 401.

Robert J. McEliece, James B. Shearer. (1978). A property of Euclid's Algorithm and an application to Pade approximation. *SIAM Jl appl. Math*. 34(4). P. 611-615.

Table 1. CPU Time to compute (7/7) matrix Pade for the semi normal power series (7) at $x = 1$ for various order of square matrices.

Matrix order	100	200	300	400	500	600	700	800	900
CPU Time (Seconds)	1	7	29	61	162	336	672	1048	1662

Table 2. CPU Time to compute (7/7) matrix Pade for the semi normal power series (7) at $x = 1$ for various order of the triangular matrices in rectangular full Packed format.

Matrix order	100	200	300	400	500	600	700	800	900
CPU Time (Seconds)	1	2	12	48	84	167	410	474	667

Table 3. Memory allocation of various matrix polynomials which are used to compute (M/N) matrix Pade approximants in square matrix case.

S.No	Name of the matrix polynomial	Memory Needed to 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 of various subroutines which are used to compute (M/N) matrix Pade approximants in square matrix case.

S.No	Name of the Subroutines	Memory needed to execute the subroutines
1	Matrix Polynomial division	$7\left((M+N+2)n^2+M+N+2\right)$
2	Matrix Polynomial multiplication	$6\left((M+N+2)n^2+M+N+2\right)$
3	Matrix Polynomial addition	$\left((M+N+2)n^2+M+N+2\right)$
4	Matrix Polynomial subtraction	$2\left((M+N+2)n^2+M+N+2\right)$
5	Matrix multiplication	n^2
6	Matrix inversion*	$6n^2$
7	Determinant	n^2

∗ Note2: We have used L U decomposition method to find inverse of a matrix (Jain, M.K., Iyengar, S.R.K.,Jain, R.K. 2003).

Table 5. Memory allocation of various polynomials which are used to compute matrix (M/N) Pade approximants in RFP format.

S.No	Name of the matrix polynomial	Memory Needed to the matrix polynomial
1	[RFP_$S(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
2	[RFP_$a(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
3	[RFP_$b(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
4	[RFP_dummy1(x)]	$(M+N+2)(n(n+1)/2)+M+N+2$
5	[RFP_dummy2(x)]	$(M+N+2)(n(n+1)/2)+M+N+2$
6	[RFP_$Q(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
7	[RFP_$R(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$
8	[RFP_$R1(x)$]	$(M+N+2)(n(n+1)/2)+M+N+2$

Table 6. Memory allocation of various subroutines that are required to compute (M/N) matrix Pade approximants in RFP format.

S.No	Name of the Subroutines	Memory needed to execute the subroutines
1	Rect_ packpoly _ division	$7(M+N+2)\,(n(n+1)/2)+7\,(M+N+2)$
2	Rect_ packpoly_ multiply	$6(M+N+2)\,(n(n+1)/2)+6\,(M+N+2)$
3	Rect_ packpoly_ addition	$(M+N+2)\,(n(n+1)/2)+(M+N+2)$
4	Rect_packpoly _ subtraction	$2(M+N+2)\,(n(n+1)/2)+2\,(M+N+2)$
5	Rect_pack_ Matrix_ multiply	$(n(n+1)/2)$
6	Rect_pack_Matrix _inversion	$(n(n+1)/2)$
7	Rect_pack	$(n(n+1)/2)$
8	Rect_ polynomial packing	$(M+N+2)\,(n(n+1)/2)+(n(n+1)/2)$

$$A = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix}$$

Figure 1. Breaking of lower triangular matrix A.

Triangular matrix **Rectangular full packed format of Triangular matrix A**

$$A = \begin{pmatrix} a_{1,1} & 0 & 0 & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 & 0 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 & 0 & 0 \\ a_{4,1} & a_{4,2} & a_{4,3} & \mathbf{a}_{4,4} & 0 & 0 \\ a_{5,1} & a_{5,2} & a_{5,3} & \mathbf{a}_{5,4} & \mathbf{a}_{5,5} & 0 \\ a_{6,1} & a_{6,2} & a_{6,3} & \mathbf{a}_{6,4} & \mathbf{a}_{6,5} & \mathbf{a}_{6,6} \end{pmatrix} \qquad \begin{pmatrix} \mathbf{a}_{4,4} & \mathbf{a}_{5,4} & \mathbf{a}_{6,4} \\ a_{1,1} & \mathbf{a}_{5,5} & \mathbf{a}_{6,5} \\ a_{2,1} & a_{2,2} & \mathbf{a}_{6,6} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \\ a_{5,1} & a_{5,2} & a_{5,3} \\ a_{6,1} & a_{6,2} & a_{6,3} \end{pmatrix}$$

$$A = \begin{pmatrix} a_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 & 0 & 0 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 & 0 & 0 & 0 \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & 0 & 0 & 0 \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & \mathbf{a}_{5,5} & 0 & 0 \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & \mathbf{a}_{6,5} & \mathbf{a}_{6,6} & 0 \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & \mathbf{a}_{7,5} & \mathbf{a}_{7,6} & \mathbf{a}_{7,7} \end{pmatrix} \qquad \begin{pmatrix} a_{1,1} & \mathbf{a}_{5,5} & \mathbf{a}_{6,5} & \mathbf{a}_{7,5} \\ a_{2,1} & a_{2,2} & \mathbf{a}_{6,6} & \mathbf{a}_{7,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & \mathbf{a}_{7,7} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} \end{pmatrix}$$

Figure 2. Rectangular full packed format of a lower triangular matrix A when $n = 6, 7$.

Matrix 1	Matrix 2	Matrix 3	Matrix 4	Matrix 5	Matrix 6	Matrix 7

Figure 3. Packing of 'm' coefficient matrices of $[S(x)]$ in rectangular full packed format ($m = 7$).

$$A = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix} \qquad D = \begin{pmatrix} D_{11} & 0 \\ D_{21} & D_{22} \end{pmatrix}$$

Figure 4. $2-$ by -2 Block form of A and D.

$$\begin{pmatrix} C_{11} & 0 \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & 0 \\ B_{21} & B_{22} \end{pmatrix}$$

Figure 5. Multiplication of A and B in $2-$ by -2 Block form.

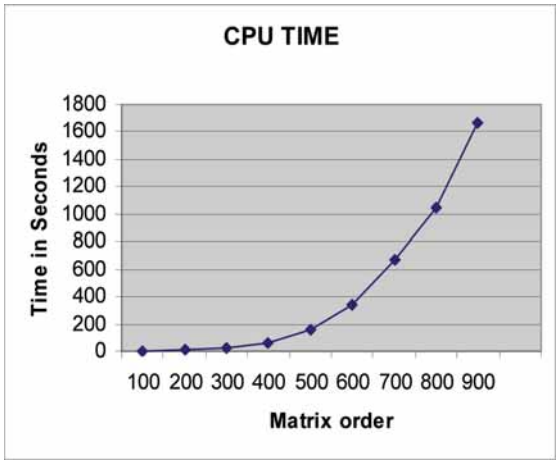

Figure 6. Graphical representation of Table 1.

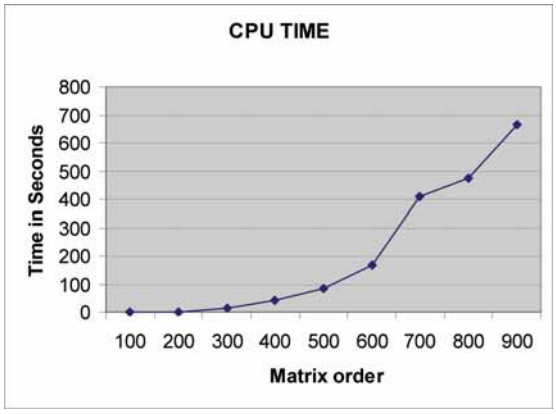

Figure 7. Graphical representation of Table 2.

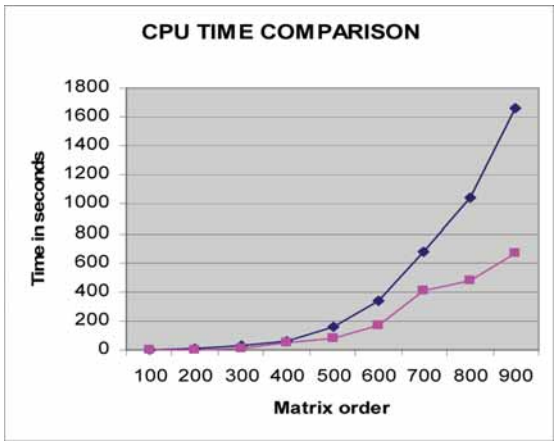

Figure 8. Graphical representation for the comparison of Table 1 and Table 2.