2008년 2학기 HW#4

수치해석기초

HW#4 : Approximation of Functions

원자핵공학과
2003-12491
이 원 재

A. Best m-th order polynomial fit

1. Write a (MATLAB) function that determines those m+1 points within the interval

In file chebypoints.m

```
function [arrayPoints] = chebypoints(a,b,m)
%%function [arrayPoints] = chebypoints(a,b,m)
%%get interval (a,b) and (m+1)interpolation points
%%return the m+1 points
zi=zeros(1,m+1);
for (i=0:m)
    xi=cos((2*i+1)*pi/(2*(m+1)));
    zi(m-i+1)=((b-a)*xi+a+b)/2;
end
arrayPoints=zi;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

위의 xi 변수는 m+1차 Chebyshev 다항식의 근으로서 [-1, 1]사이에 존재하는데 이를 [a, b] 구간의 분점으로 변환시키기 위해 zi 변수를 도입하였다. 근이 작은 값부터 차례대로 나오도록 zi 벡터에는 뒤에서부터 대입하였다.

2. Obtain the m+1 points and the function values at those m+1 points for

a. $f(x) = xe^x$ **in [0, 1.5] with m=4**

b. $f(x) = \cos(\ln x)$ **in [1, 3] with m=5**

이 문제를 풀기 위해 다음과 같은 세 개의 matlab 함수를 작성한다.

In file func2a.m

```
function [y] = func2a(x)
%%function [y] = func2a(x)
%%take length of x
%%return function value of y=x*exp(x)
n=length(x);
for (i=1:n)
    y(i)=x(i)*exp(x(i));
end
%%%%%%%%%%%%%%%%%%%%
```

In file func2b.m

```
function [y] = func2b(x)
%%function [y] = func2a(x)
%%take length of x
%%return function value of y=cos(ln(x))
n=length(x);
for (i=1:n)
    y(i)=cos(log(x(i)));
end
%%%%%%%%%%%%%%%%%%%%
```

In file hw4a2.m

```
function [xa,ya,xb,yb] = hw4a2()
% A. 2번 문제를 풀기위한 루틴으로서
% 문제에서 요구하는 구간과 차수에 대해
% a번과 b번 함수에 대한 chebyshev 분점들과 함수값을 리턴한다.
xa=chebypoints(0,1.5,4);
[ya]=func2a(xa);
xb=chebypoints(1,3,5);
[yb]=func2b(xb);
```

이제 Matlab command window에서 다음과 같은 함수호출을 하면

```
>> [xa,ya,xb,yb]=hw4a2()
xa =
    0.0367    0.3092    0.7500    1.1908    1.4633

ya =
    0.0381    0.4212    1.5878    3.9177    6.3217

xb =
    1.0341    1.2929    1.7412    2.2588    2.7071    2.9659

yb =
    0.9994    0.9672    0.8501    0.6860    0.5438    0.4650
>>
```

xa는 [0, 1.5] 의 m=4 에 해당하는 m+1 개의 분점들에 해당하고 ya 는 해당하는 분점들에서 $f(x) = xe^x$ 의 함수값에 해당한다. xb 와 yb 도 각각의 구간과 차수에 해당하는 분점과 함수값이다.

3. Determine the (m+1) coefficient of each term appearing in the Lagrange interpolation (Use your own program or the MATLAB script provided – lagp0.m)

먼저 다음과 같은 함수를 작성한다.

In file hw4a3.m

```
function [coeff_a,coeff_b] = hw4a3()
% a, b번 함수에 대한 구간에서의 분점과 그 분점들에서의
% 함수값을 이용해 Lagrange 계수를 구한다.
xa=chebypoints(0,1.5,4);
[ya]=func2a(xa);
coeff_a=cifi(xa,ya);
xb=chebypoints(1,3,5);
[yb]=func2b(xb);
coeff_b=cifi(xb,yb);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이 함수는 내부에서 chebypoints.m , func2a.m , func2b.m 그리고 예전에 작성했던 cifi.m의 함수를 호출한다. 이렇게 구한 Lagrange 계수를 벡터 형태로 리턴한다.

Matlab command window에서 다음과 같이 함수 호출을 실행하여 m+1개의 계수를 얻을 수 있다.

```
>> [coeff_a, coeff_f]=hw4a3()
coeff_a =
    0.1190   -3.4460   16.0578  -32.0546   19.7569
coeff_f =
   -1.3796    3.6475   -4.3795    3.5339   -2.0507    0.6418
>>
```

4. Divide the interval into 1000 subintervals and evaluate the polynomial and the function at 1001 points and then determine the maximum absolute error you're your own or lagp.m).

다음과 같은 함수를 작성한다.

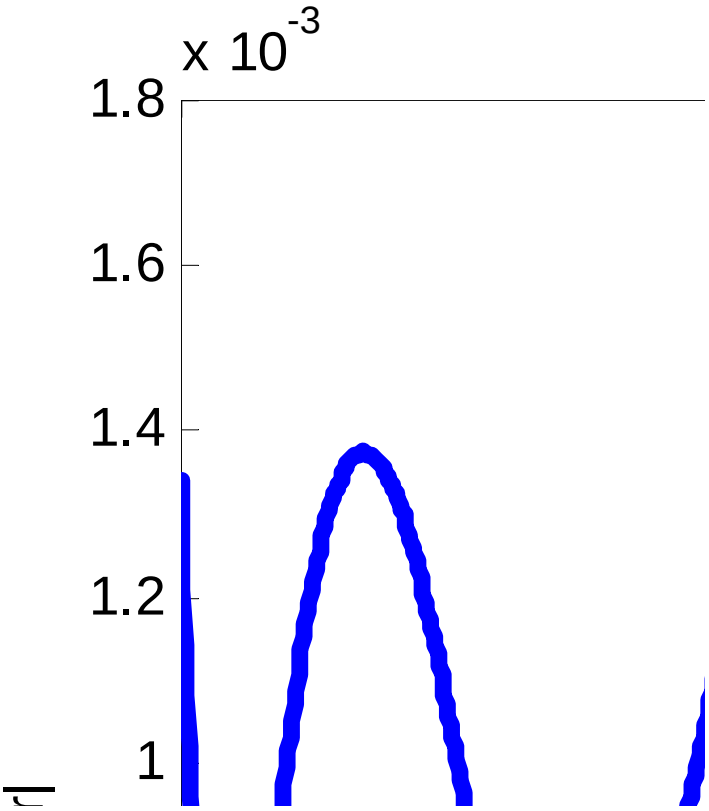
In file hw4a4a.m

```
function [maxErr] = hw4a4a(a,b,m,sub)
% 함수 y=x*exp(x) 에 대해서
% 구간, [a,b]와 차수 m 그리고 subinterval의 개수 sub를 argument로 받는다.
% Chebyshev polynomial과 일치시킨 라그랑지 내삽법을 수행하여
% 각 subinterval point에서 p(x)값을 얻고 실제 함수 값과 비교 하여 오차를 얻어
% 최대 오차를 리턴하고, 오차의 크기를 그래프로 나타낸다.
increment=(b-a)/sub;
xsubs=a:increment:b;                    %subinterval의 격자점을 구한다.
xCheby=chebypoints(a,b,m); %인자로 받은 구간을 Chebyshev 함수의 근으로
                                        %나타낸다.
[y]=func2a(xCheby);                     %각 분점에서의 함수값을 취한다.
[cf]=cifi(xCheby,y);                    %분점과 분점에서의 함수값을 이용하여 계수를 구한다.
n=length(xsubs);                        %모든 subinterval+1의 격자점에 대해서
for (k=1:n)
    approxF(k)=plintp(xsubs(k),xCheby,cf);      %approximation 함수값과
    realF(k)=func2a(xsubs(k));                  %실제 함수값을 구한다.
end
maxErr=0.0;                             %초기 오차는 0으로 하고
for (k=1:n)
    tmpErr(k)=abs( (approxF(k)-realF(k)) );     %오차의 절대값을 임시저장한 후
    if (tmpErr(k)>=maxErr)                      %지금까지의 최대오차보다 크면
        maxErr=tmpErr(k);                       %최대오차를 업데이트한다.
    end
end
plot(xsubs,tmpErr,'LineWidth',3);       %오차의 크기를 그래프로 나타낸다.
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

매트랩 커맨드 윈도우에서 다음과 같은 명령어를 입력한다.

```
>> maxErr_a=hw4a4a(0,1.5,4,1000)
maxErr_a =
    0.0018
>>
```



그래프를 보면 각 분점에 대해서는 오차가 매우 적은 반면에 분점의 사이에서는 오차가 0.0018보다 작게 나오고 있음을 알 수 있다. 그리고 이 오차는 1/2^m-1 보다 작다는 것을 확인 할 수 있다.

b)번 함수에 대해서도 마찬가지로 하되 내부에서 함수호출 하는 부분만 func2b()로 바꿔주면 된다.

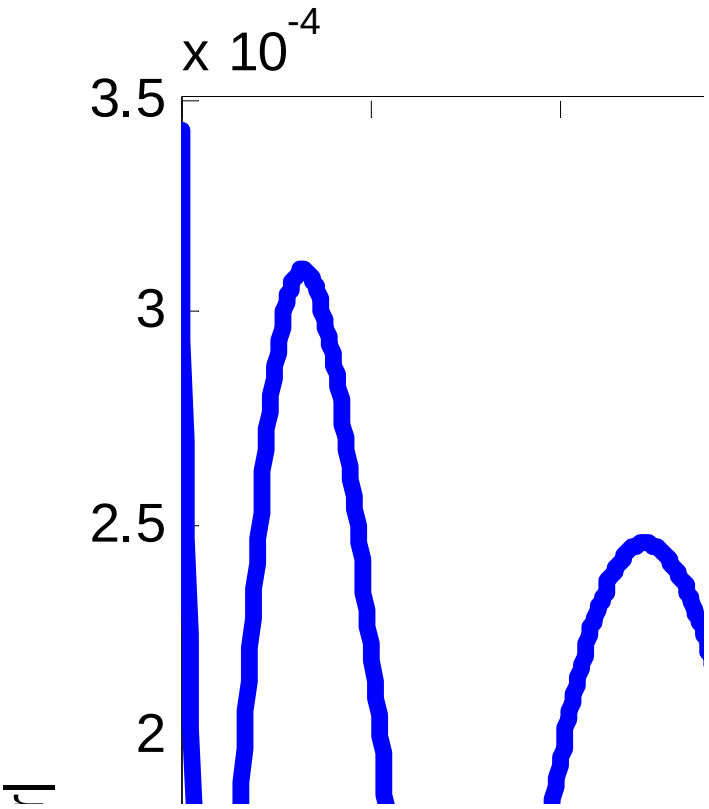
```
function [maxErr] = hw4a4b(a,b,m,sub)
.....
[y]=func2b(xCheby);                    %각 분점에서의 함수값을 취한다.
[cf]=cifi(xCheby,y);                   %분점과 분점에서의 함수값을 이용하여 계수를 구한다.
n=length(xsubs);                       %모든 subinterval+1의 격자점에 대해서
for (k=1:n)
    approxF(k)=plintp(xsubs(k),xCheby,cf);      %approximation 함수값과
    realF(k)=func2b(xsubs(k));                  %실제 함수값을 구한다.
end
....
```

이제 매트랩 커맨드 윈도우에서 다음을 입력하면

```
>> maxErr_b=hw4a4b(1,3,5,1000)
```

```
maxErr_b =
   3.4210e-004

>>
```



최대 오차는 x=1 일 때 3.4210e-004 로 나오며 앞에서 수행한 것보다 차수가 하나 더 많은 만큼 오차가 2배이상 줄어든 것을 볼 수 있다.

```
>> maxErr_b=hw4a4b(1,3,4,1000)
maxErr_b =
      0.0014
>>
```

다음과 같이 차수를 5에서 4로 줄여보면 오차가 0.0014로 앞에서 수행한 a)번 함수에 대한 4차의 오차 0.0018과 비슷한 것을 볼 수 있다. 또한 m을 차례대로 올려가거나 내려감에 따라 최대오차는 약 4배로 변하는 것을 확인 할 수 있다. 즉, 한 차수씩 감소시켜 감에 따라 최대오차가 4배로 변해가는 것을 확인 할 수 있다.

m=6 일 때 최대 오차 ---> 8.0017e-005
m=5 일 때 최대 오차 ---> 3.4210e-004
m=4 일 때 최대 오차 ---> 0.0014
m=3 일 때 최대 오차 ---> 0.0059
m=2 일 때 최대 오차 ---> 0.0220

5. Choose the m+1 points in the following ways to determine the m-th order polynomial. Compare then the maximum error of each case with the one obtained from 4 to show that the Chebyshev polynomial based points give indeed the lowest maximum error.

a. Equidistance

b. 10 Different sets of random points (use the rand function-rand(m+1,1))

다음과 같은 함수는 함수 a)에 대해서 Chebyshev 함수의 근으로 하는 구간, m등분점, 10개 세트의 무작위수 구간으로 구한 오차를 그래프와 함께 리턴한다.

In file hw4a5a.m

```
function [errCheby,errEqui,errRand] = hw4a5a(a,b,m,sub)
increment=(b-a)/sub;
xsubs=a:increment:b;
%%%%%%%%%%% Chebyshev Polynomial 방법으로 최대오차를 최소화 시키는 부분
xCheby=chebypoints(a,b,m);
[y]=func2a(xCheby);
[cf]=cifi(xCheby,y);
n=length(xsubs);
for  (k=1:n)
    approxF(k)=plintp(xsubs(k),xCheby,cf);
        realF(k)=func2a(xsubs(k));
end
maxErr=0.0;
for  (k=1:n)
    tmpErr1(k)=abs(  (approxF(k)-realF(k))  );
        if  (tmpErr1(k)>=maxErr)
            maxErr=tmpErr1(k);
        end
end
errCheby=maxErr;
%%%%%%%%%%% 구간을 등분하는 분점을 사용하였을때 오차를 구하는 부분
incre=(b-a)/m;
xEqui=a:incre:b;
[y]=func2a(xEqui);
[cf]=cifi(xEqui,y);
n=length(xsubs);
for  (k=1:n)
    approxF(k)=plintp(xsubs(k),xEqui,cf);
        realF(k)=func2a(xsubs(k));
end
maxErr=0.0;
for  (k=1:n)
```

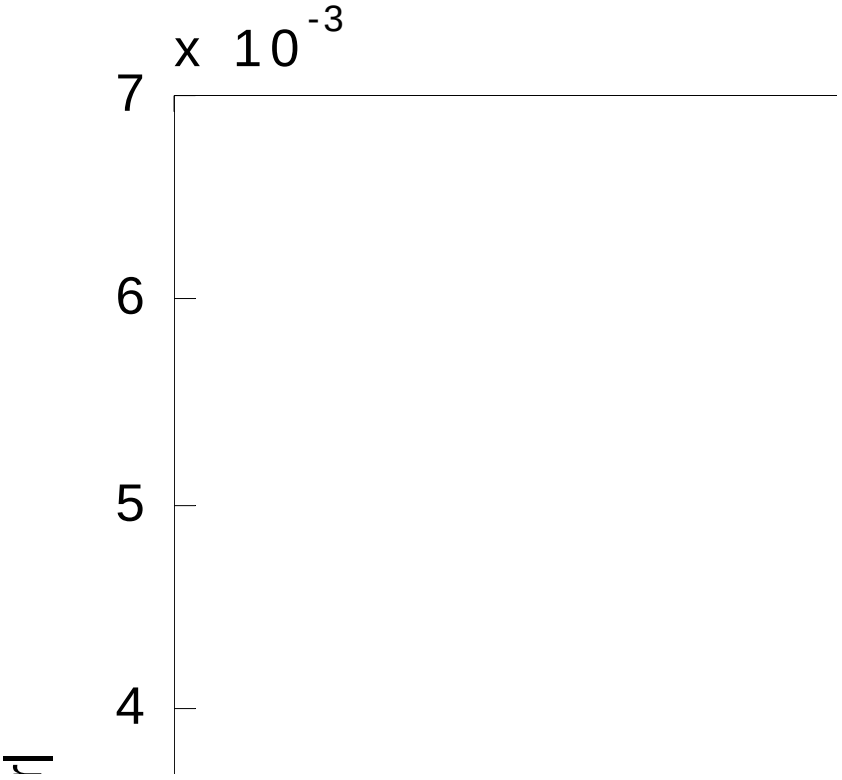
```
    tmpErr2(k)=abs(  (approxF(k)-realF(k))  );
          if  (tmpErr2(k)>=maxErr)
              maxErr=tmpErr2(k);
          end
end
errEqui=maxErr;
%%%%%%%% m+1개의 무작위 수를 갖는 10개의 set에서 각각에 대해 오차 구하는 부분
for  (j=1:10)
    increment=rand(m+1,1);
    for  (i=1:m+1)
              xRand(i)=a+increment(i)*(b-a);
    end

    [y]=func2a(xRand);
    [cf]=cifi(xRand,y);
    n=length(xsubs);
    for  (k=1:n)
        approxF(k)=plintp(xsubs(k),xRand,cf);
              realF(k)=func2a(xsubs(k));
    end
    maxErr=0.0;
    for  (k=1:n)
        tmpErr3(j,k)=abs(  (approxF(k)-realF(k))  );
              if  (tmpErr3(j,k)>=maxErr)
                  maxErr=tmpErr3(j,k);
              end
    end
    errRand(j)=maxErr;
end
                      %일단 첫번째로 구한 랜덤 구간에 대해서 그래프 그림.
plot(xsubs,tmpErr1,'r:',xsubs,tmpErr2,xsubs,tmpErr3(1,:),'LineWidth',3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

위 함수를 다음과 같이 호출하면

```
>>  [c  e  r]=hw4a5a(0,1.5,4,1000)
c  =
     0.0018
```

```
e =
    0.0032
r =
    0.0061    0.0390    0.0314    0.0287    0.0381    0.0800    0.2025    0.0112    0.0177
0.0244
>>
```

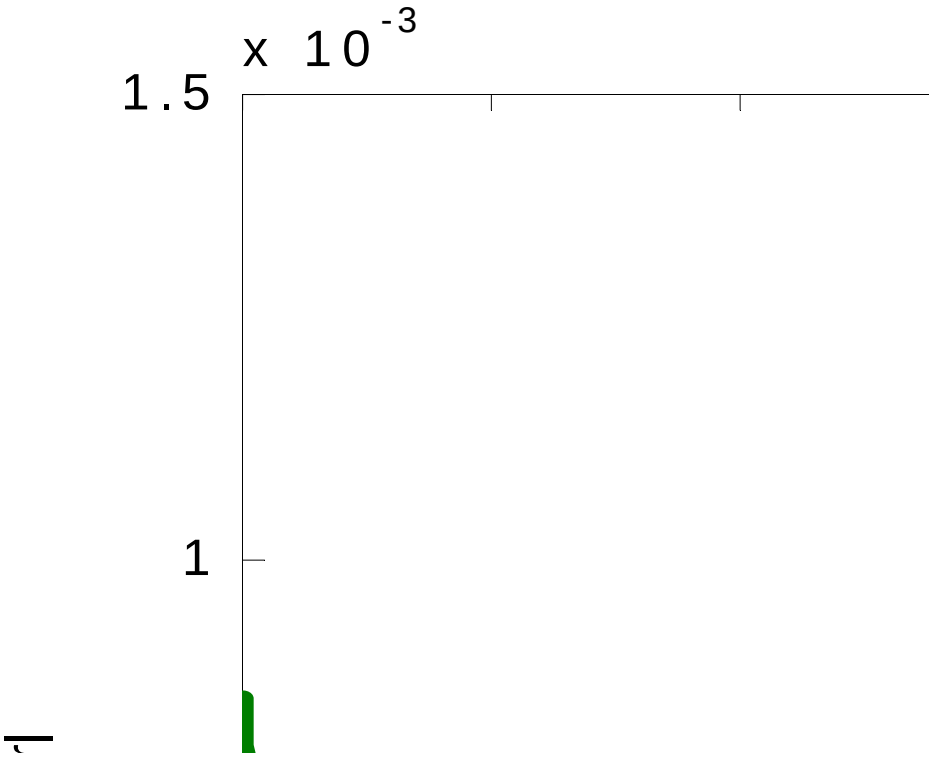


Chebyshev Polynomial의 근에 맞춘 경우 오차가 0.0018로 가장 작게 나타났으며 등간격으로 한 경우 0.0032, 그리고 무작위 수로 한 경우는 0.0061에서 0.2025까지 다양한 최대오차가 나타났다. 그래프에서 보면 random 세트에서 나온 최소오차로 그래프가 그려졌음에도 불구하고 가장 큰 오차를 보이고 있으며 역시 Chebyshev의 근으로 구한 분점을 통해 구한 Lagrange interpolation의 최대오차가 가장 작음을 확인 할 수 있다. 등간격으로 한 경우 x가 전체 구간의 중심인 0.75에 가까운 구간에서는 Chebyshev 법으로 구한 경우보다 오차가 더 작지만 x가 0.75로부터 멀어짐에 따라 오차가 커지는 것을 확인 할 수 있다.

이제 위의 함수 hw4a5a.m 안에서 함수 호출부분인 func2a() 부분만 func2b()로 바꿔서 hw4a5b.m으로 저장하여 같은 과정을 수행하면 다음과 같다.

matlab command window에서

```
>> [c e r]=hw4a5b(1,3,5,1000)
c =
   3.4210e-004
e =
   6.7387e-004
r =
    0.0015    0.0036    0.0041    0.0063    0.0064    0.0029    0.0119    0.0128    0.0037
0.0009
>>
```



역시 Chebyshev로 구한 경우의 최대오차가 최소임을 확인 할 수 있다. 무작위 수로 구한 구간에서는 구간 간격이 좁은 경우 작은 오차를 보이나 구간 간격이 큰 지점에서는 오차가 매우 커지는 것을 확인 할 수 있다. 또한 이 경우에도 등간격으로 구한 경우에 x가 전체 구간의 중심인 2에 가까운 구간에서는 오차가 작으나, 여기에서 멀어질수록 오차가 커지는 것을 확인 할 수 있다.

B. Pade Approximation of Matrix Exponential

2. Use different pairs of (n,m) and observe the error of Pade approximation of the matrix exponential until you obtain the error in each component of the matrix exponential less than 0.5%. In MATLAB, matrix exponential can be readily obtained by expm(A).

다음과 같은 함수를 작성한다.

In file exppadematrix.m

```
function [percentErr,padeValue,funcValue] = exppadematrix(n,m,A)
%% 입력 파라메터로 분자의 차수와 분모의 차수 그리고 행렬 A를 받는다
%% exppade()를 호출하여 각 계수 벡터를 넘겨받고
%% 행렬 A에 대한 exp 의 근사를 계산하여
%% matlab 내부 함수인 expm()과 비교하여 오차를 퍼센트로 나타낸다.
%% 출력 스트림은 첫번째 항이 퍼센터 오차, 둘째항이 pade approximation으로 얻은 행렬
%% 세번째 항은 내부 함수로 얻은 expm(A) 값이다.
[an,bm]=exppade(n,m);                    % 제공받은 함수의 호출
tmpnume=zeros(size(A));
tmpdeno=zeros(size(A));
for (i=0:n)
    tmpnume=tmpnume+an(i+1)*A^i;
end
for (i=0:m)
    tmpdeno=tmpdeno+bm(i+1)*A^i;
end
padeValue=tmpnume/tmpdeno;
funcValue=expm(A);

percentErr=abs( (padeValue-funcValue)./funcValue *100);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

물론 이 함수를 사용하기 위해서는 같은 디렉토리에 제공된 함수 exppade.m이 있어야 한다.

그리고 n,m 값을 변화시켜가며 오차를 구해서 0.5%이하인 경우에 해당하는 n,m 값을 알아내기 위해 다음과 같은 함수를 작성한다.

In file hw4b2.m

```
function [result,nm] = hw4b2(A)
%% 변화하는 n, m 값에 대해서 pade approximation 의 오차 변화를 살피기 위해
%% n,m값을 0부터 7까지 변화시켜 가면서 각 항의 오차를 저장한다.
```

```
%% 각 n,m에 대해 오차가 0.5% 보다 작은 경우만 골라내서 nm 변수에 저장하여 리턴한다.
nbound=7;
mbound=7;
nm=NaN;
i=1;
for (n=0:nbound)
    for (m=0:mbound)
        errMatrix=exppadematrix(n,m,A);
            result(i,1)=n;
            result(i,2)=m;
            result(i,3)=errMatrix(1,1);
            result(i,4)=errMatrix(1,2);
            result(i,5)=errMatrix(2,1);
            result(i,6)=errMatrix(2,2);
            i=i+1;
        end
end
len=i-1;

j=1;
for (i=1:len)
    if (result(i,3)<0.5 &&result(i,4)<0.5 &&result(i,5)<0.5 &&result(i,6)<0.5 )
        nm(j,1)=result(i,1);
        nm(j,2)=result(i,2);
        j=j+1;
    end
end
nm=nm';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이제 매트랩 커맨드 윈도우에서 다음과 같은 명령어를 입력하면

```
>> A=[1,0.5;-0.5,3]

A =
    1.0000    0.5000
   -0.5000    3.0000
```

```
>> B=[2,1;-1,5]
B =
      2       1
     -1       5

>> [res_A,nm_A]=hw4b2(A);
>> [res_B,nm_B]=hw4b2(B);
>> nm_A

nm_A =
Columns 1 through 22
2   3   3   3   4   4   4   4   4   5   5   5   5   5   5   6   6   6   6   6   6   6
7   5   6   7   3   4   5   6   7   2   3   4   5   6   7   1   2   3   4   5   6   7

Columns 23 through 29
7   7   7   7   7   7   7
1   2   3   4   5   6   7

>> nm_B

nm_B =
5     5     6     6     6     7     7     7     7     7
6     7     5     6     7     3     4     5     6     7

>>
```

위의 nm_A, nm_B 행렬은 각 column이 주어진 행렬 A, B에 대한 pade approximation의 오차가 0.5% 미만인 분자, 분모 차수쌍을 의미한다. 즉 (n,m) 쌍이 (2,7), (3,5), (3,6)... 등에서 행렬 A가 오차 0.5% 미만으로 exponential 값이 계산될 수 있는 것을 의미한다. 행렬 B에 대해서는 (n,m) 쌍이 (5,6), (5,7),.. 일 때 비교적 작은 오차에서 exponential 값을 계산할 수 있다.

3. Matrix B would require higher orders. What would be the major factor requiring higher orders in Pade approximation of matrix exponential?

매트릭스 B는 가능한 n과 m을 0부터 7까지로 했을 때 (5,6)부터 나타나는 반면에 매트릭스 A는 (2,7), 과 (3,5)부터 나타나고 있음을 확인할 수 있는 데, 이는 Pade approximation이 매클로린 급수의 n+m차 항까지 유리함수를 일치시키는 데에서 기인한다. 즉, 매클로린 급수는 0을 중심으로 한 테일러 전개와 같으므로 주어진 함수에 값을 넣을때에 0에서 멀리 떨어질수록 오차가 커져 정해진 오차율(0.5%)내로 들어오려면 더 큰 차수의 유리함수가 필요하게 되는 것이다. 이를 확인하기 위해 hw4b2.m을 수정하여 매트릭스뿐만 아니라 일반 실수도 받을 수 있도록 고친 hw4b3.m을 작성한다.

In file hw4b3.m

```
function [result,nm] = hw4b3(A)
%% 변화하는 n, m 값에 대해서 pade approximation 의 오차 변화를 살피기 위해
%% n,m값을 0부터 7까지 변화시켜 가면서 각 항의 오차를 저장한다.
%% 각 n,m에 대해 오차가 0.5% 보다 작은 경우만 골라내서 nm 변수에 저장하여 리턴한다.
%% A의 크기에 따라 자동으로 총 element의 개수를 취하여 hw4b2와 같은 결과를 낸다.
nbound=7;
mbound=7;
[a,b]=size(A);
dimension=a*b;
nm=NaN;

i=1;
for (n=0:nbound)
    for (m=0:mbound)
        errMatrix=exppadematrix(n,m,A);
                    result(i,1)=n;
                    result(i,2)=m;
                    for (k=1:dimension)
                        result(i,2+k)=errMatrix(k);
                    end
                    i=i+1;
          end
end
len=i-1;

j=1;
for (i=1:len)
    tmp=1;
          mask=0;
```

```
            for  (k=1:dimension)
        mask=result(i,2+k)<0.5;
                    tmp=tmp*mask;
    end
    if  (tmp==1)
        nm(j,1)=result(i,1);
                    nm(j,2)=result(i,2);
                    j=j+1;
    end
end
nm=nm';
```

%%%%%%%%%%%%%%%%%%%%%%%%

이 프로그램을 사용하여 다음을 확인해보면

```
>>[res_1,nm_1]=hw4b3(0);nm_1

nm_1 =

   Columns 1 through 22

     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1     2     2     2     2     2     2
     0     1     2     3     4     5     6     7     0     1     2     3     4     5     6     7     0     1     2     3     4     5

   Columns 23 through 44

     2     2     3     3     3     3     3     3     3     3     4     4     4     4     4     4     4     4     5     5     5     5
     6     7     0     1     2     3     4     5     6     7     0     1     2     3     4     5     6     7     0     1     2     3

   Columns 45 through 64

     5     5     5     5     6     6     6     6     6     6     6     6     7     7     7     7     7     7     7     7
     4     5     6     7     0     1     2     3     4     5     6     7     0     1     2     3     4     5     6     7

>>[res_1,nm_1]=hw4b3(1);nm_1

nm_1 =

   Columns 1 through 22

     0     0     0     1     1     1     1     1     2     2     2     2     2     2     3     3     3     3     3     3     3     4
     5     6     7     3     4     5     6     7     2     3     4     5     6     7     1     2     3     4     5     6     7     0
```

Columns 23 through 44

```
4     4     4     4     4     4     4     5     5     5     5     5     5     5     5     6     6     6     6     6     6     6
1     2     3     4     5     6     7     0     1     2     3     4     5     6     7     0     1     2     3     4     5     6
```

Columns 45 through **53**

```
6     7     7     7     7     7     7     7     7
7     0     1     2     3     4     5     6     7
```

```
>> [res_1,nm_1]=hw4b3(2);nm_1
```

nm_1 =

Columns 1 through 22

```
1     1     2     2     2     2     3     3     3     3     3     4     4     4     4     4     4     5     5     5     5     5
6     7     4     5     6     7     3     4     5     6     7     2     3     4     5     6     7     1     2     3     4     5
```

Columns 23 through **40**

```
5     5     6     6     6     6     6     6     6     6     7     7     7     7     7     7     7     7
6     7     0     1     2     3     4     5     6     7     0     1     2     3     4     5     6     7
```

```
>> [res_1,nm_1]=hw4b3(3);nm_1
```

nm_1 =

Columns 1 through **22**

```
2     3     3     3     4     4     4     4     4     5     5     5     5     5     5     6     6     6     6     6     6     6
7     5     6     7     3     4     5     6     7     2     3     4     5     6     7     1     2     3     4     5     6     7
```

Columns 23 through 29

```
7     7     7     7     7     7     7
1     2     3     4     5     6     7
```

```
>> [res_1,nm_1]=hw4b3(5);nm_1
```

nm_1 =

```
5     5     6     6     6     6     7     7     7     7     7
6     7     4     5     6     7     3     4     5     6     7
```

이를 통해 x 값이 0에서 멀어질수록 더 높은 차수를 요구하는 것을 알 수 있다.
따라서 이것을 행렬에 확장시켜 생각해 보면 행렬의 행렬식이 0에서 멀어질수록 더 높은 차수를 요구할 것으로 생각되는 데 , A와 B의 행렬식을 확인해 보면

```
>> det(A)
ans =
    3.2500
>> det(B)
ans =
    11
>>
```

로서 과연 B의 determinent가 더 큼을 알 수 있다.

예를 들어 determinent가 작은 어떤 행렬에 대해 조사해 보면

```
>> C=[1,0.1;0.1,1]

C =

    1.0000    0.1000
    0.1000    1.0000

>> [res_1,nm_1]=hw4b3(C);nm_1

nm_1 =

  Columns 1 through 22

0    0    1    1    1    1    2    2    2    2    2    3    3    3    3    3    3    4    4    4    4    4
6    7    4    5    6    7    3    4    5    6    7    2    3    4    5    6    7    1    2    3    4    5

  Columns 23 through 44

4    4    5    5    5    5    5    5    5    5    6    6    6    6    6    6    6    6    7    7    7    7
6    7    0    1    2    3    4    5    6    7    0    1    2    3    4    5    6    7    0    1    2    3

  Columns 45 through 48

7    7    7    7
4    5    6    7
```

위와 같이 determinent가 작으면 더 작은 차수로도 Pade 근사를 잘 할 수 있음을 알 수 있다.

「부록」 function list

chebypoints.m

func2a.m

func2b.m

hw4a2.m

hw4a3.m

hw4a4a.m

hw4a4b.m

hw4a5a.m

hw4a5b.m

exppade.m

exppadematrix.m

hw4b2.m

hw4b3.m

cifi.m

plintp.m