2008년 2학기 HW#5

수치해석기초

HW#5 : 1-D Particle Diffusion and LU Factorization

원자핵공학과
2003-12491
이 원 재

Part 1. 1-D Particle Diffusion

A. Uniform Source

1. Let $n = 10 \cdot 2^k$ be the total number of meshes at the k-th step with k=0 being the base step. Write a MATLAB (or any other language) program to construct and solve the linear system for the discretized equations with k as a parameter. Use the normal form of Eq. (1) for discretization.

In file diffusolve.m

```
function [phi] = diffusolve(k)
%% k를 parameter로 받아서 메쉬의 개수를 결정하고
%% 주어진 geometry와 조건수에 대해서 확산방정식을 풀어
%% 그 해를 리턴한다. 해 벡터의 크기는 매쉬의 개수와 같다.
%%%%%%%%%%%%%%%%%%%%%%parameters%%%%%%%%%%%%%%%%%%%%
a=10;
D=2.0;
sig=0.5;

n=10*2^k;%메쉬의 개수
h=a/n;%잘게 자른 구간 폭
h2=h*h; %연산수를 줄이기 위해 미리 제곱 계산해서 변수에 저장

sigtilda=sig/D;

swc=1; %source switcher로서 1이면 s(x)=1 이고
       %1이 아닌 다른 수이면 s(x)=1-0.2|x-5|
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A=zeros(n); %n x n 매트릭스 생성
b=zeros(n,1); %source 저장용 매트릭스 생성

A(1,1)=1/h2+sigtilda;          %왼쪽 경계에서 reflective 조건
A(1,2)=-1/h2;
for (i=2:n-1)
    A(i,i-1)=-1/h2;
    A(i,i)=2/h2+sigtilda;
    A(i,i+1)=-1/h2;
end
A(n,n-1)=-1/h2;
A(n,n)=3/h2+sigtilda;          %오른쪽 경계에서 null flux 조건
for (i=1:n)
  b(i)=sourcetilda(swc,a,n,D,i);
end
```

```
phi=A\b;   %linear system의 풀이

%%%%%%%%%%%%%%%%%%%%소스 리턴 부함수%%%%%%%%%%%%%%%%%%%%
function [ytilda] = sourcetilda(swc,a,n,D,i)
if (swc==1)
    y=1;
    ytilda=y/D;
else
    h=a/n;
    x=a*i/n;
    x=x-h/2;   %소구간의 중간점을 취하기 위해서
    y=1-0.2*abs(x-5); %문제에 주어진 텐트모양 함수
    ytilda=y/D;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

위 파일은 1차원 매쉬의 사이즈를 결정하는 입력 매개변수 k를 받아서 매쉬의 사이즈를 결정하며, 숙제에 제시된 1차원 확산방정식을 푸는 알고리즘을 담고 있다. 확산방정식의 2차 미분을 차분화하여 현재 그리드에서의 함수값을 양쪽 옆에서의 함수값과 연결시키며, 양쪽 경계에서 경계조건에 따라 2/h2에 1/h2를 더해주거나 빼서 linear system을 완성하였다. 이 시스템을 풀어서 나오는 해(매쉬 개수만큼)를 벡터로 리턴한다. 한편 소스텀에서 문제에 주어진 텐트모양의 함수도 다룰 수 있도록 subfunction을 마련하고 스위칭할 수 있는 인자를 받아 1을 리턴하거나 각 구간에서의 임의의 함수값을 리턴할수 있게 하였다.

2. Define a vector consisting of the ten subdomain fluxes and determine the vector elements from the solution at the k-th step. To obtain the subdomain flux, you need to sum up the fluxes belonging to each subdomain and multiply the sum by the mesh size. Also obtain the total flux for the entire domain. For the analytic solution, define the subdomain vector in the same way, namely the summation of the fluxes obtained at the same discretized points as the numerical solution. Do not bother with integration. Define the error vector as the difference between the numerical subdomain flux vector and the analytic subdomain flux vector. And define the total flux error as the difference between the numerical total flux and the analytic total flux. Then observe the error reduction behavior at each step. See if you see indeed the error reduction by 1/4 at each step. Proceed until the error reduction ratio falls within 0.25 ± 0.0001 ? Plot the flux distribution at the base step and the final step at the same graph.

앞에서 구한 솔루션을 subdomain flux로 변환하는 프로그램을 다음과 같이 작성할 수 있다.

In file sdflux.m (SubDomain FLUX)

```
function [phisub] = sdflux(phi)
%% n=10*2^k 개의 매쉬에 대한 플럭스를
%% 10개의 sub 도메인에서의 플럭스로 나타낸다.
%%%%%%%%%%%%parameters%%%%%%%%%%%%%%%%%%%%%%
subdomain=10;  %sub domain의 개수를 10개로 줌
a=10;   %현재 푸는 문제의 구간이 10cm 로 나옴
[r,c]=size(phi);
n=r*c;   %해벡터의 크기로부터 구간을 자른 개수를 알아냄
h=a/n;  % mesh size구함 전체 구간 길이를 매쉬 개수로 나눔.

phisub=zeros(subdomain,1); %리턴할 서브 도메인 벡터의 선언

num=n/subdomain; %sub domain 하나당 들어갈 해벡터의 원소 개수 결정
%예를 들어 40개 원소로 된 해벡터가 들어오면 벡터 4개씩 잘라서 sub 도메인에 넣음

for (j=1:subdomain)
    tmpsum=0.0;
    for (k=1:num)  %하나의 subdomain에 들어갈 개수만큼 새어서 더함
        tmpsum=tmpsum+phi((j-1)*num+k);
    end
    phisub(j)=tmpsum*h;  %구간 간격을 곱해서 적분한 효과를 줌
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이제 total flux를 구하기 위해서는 입력받은 벡터의 원소들을 모두 더하고 매쉬 사이즈로 곱해주기만 하면 된다. 그러나 위의 프로그램을 사용하여 subdomain 개수를 1로 바꿔줘도 같은 결과가 나온다. 다만, 불필요한 계산을 몇 개 수행하지만 그렇게 큰 시간이 소요되지는 않으므로 위의 프로그램에서 subdomain값을 1로 바꾸고 파일이름만 바꿔서 사용하기로 한다.

In file totalflux.m

```
function [phisub] = totalflux(phi)
%% n=10*2^k 개의 매쉬에 대한 플럭스를
%% 1개의 sub 도메인에서의 플럭스로 나타낸다. 즉 토탈 플럭스를 구한다.
%%%%%%%%%%%%%%%%%%%%parameters%%%%%%%%%%%%%%%%%%%%%%%%%

subdomain=1;  %sub domain의 개수를 1개로 줌
a=10;   %현재 푸는 문제의 구간이 10cm 로 나옴
[r,c]=size(phi);
n=r*c;   %해벡터의 크기로부터 구간을 자른 개수를 알아냄
h=a/n;  % mesh size구함 전체 구간 길이를 매쉬 개수로 나눔.

phisub=zeros(subdomain,1); %리턴할 서브 도메인 벡터의 선언

num=n/subdomain; %sub domain 하나당 들어갈 해벡터의 원소 개수 결정

for (j=1:subdomain)
    tmpsum=0.0;
    for (k=1:num)
        tmpsum=tmpsum+phi((j-1)*num+k);
    end
    phisub(j)=tmpsum*h;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

앞서서 diffusion equation풀 때 10*2^k 개의 구간으로 나눠서 그만큼의 해를 냈던것과 마찬가지로 이제는 analytic solution에 대해서 같은 형태의 해를 내주는 프로그램을 다음과 같이 작성한다.

In file asolution.m (Analytic SOLUTION)

```
function [yn] = asolution(k)
% diffusion 방정식의 해석해에 대해서
% n=10*2^k 개의 구간에 대해 n개의 값을 벡터로 리턴한다
% 단, 이 경우는 constant source에 대한 경우의 해석해에 한하는 것이다.
a=10;
n=10*2^k;
h=a/n;
```

```
%%%%%%%%%%%%%%%%해석해를 위한 파라메터들%%%%%%%%%%%%
D=2.0;
s=1;
sig=0.5;
s_tilda=s/D;
sig_tilda=sig/D;
B=sqrt(sig_tilda);

for  (i=1:n)
    x=a*i/n;
    x=x-h/2;   %소구간의 중간점을 취하기 위해서
    yn(i)=s_tilda/(B*B)*(1-cosh(B*x)/cosh(B*a));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이제는 오차가 매쉬수 사이즈가 증가하여, 즉 매쉬간격의 몇차에 의존하는지 알아보기위해, 2차에 의존하여 1/4로 줄어들것으로 예상되는데 다음과 같은 프로그램은 오차의 감소율이 1/4에 0.01% 이내로 가까워지면 step 늘이는 것을 중단한다.

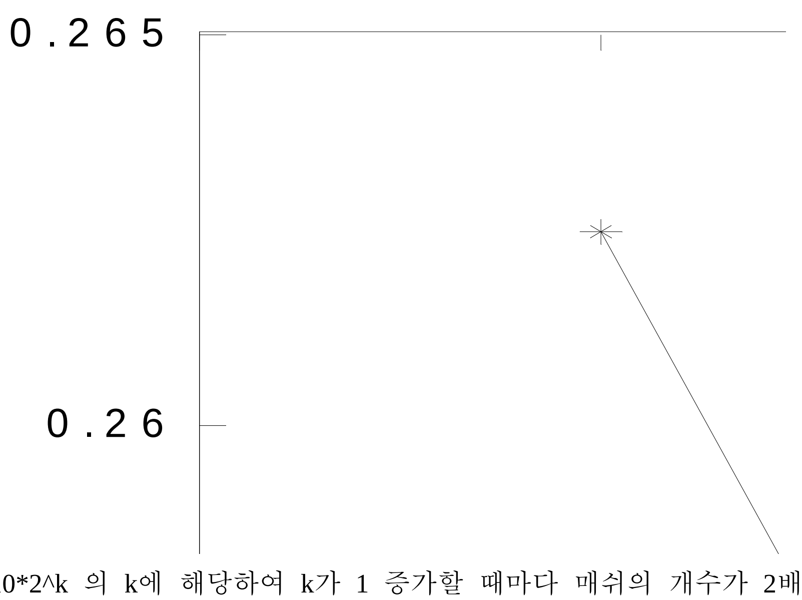
In file hw5a2.m

```
function [] = hw5a2(col)
%%관측하고 싶은 에러를 입력인자로 준다. 예를들어 total error의 reduction을 보고싶으면 0
%% i번째 subdomain 값에 대한 오차의 reduction을 알고 싶으면 i 을 인자로 주면 된다.
%% 여기서 i는 1~10
steps=7;
erd=zeros(steps+1,11);
rho=zeros(steps+1,1);
erd(:)=NaN;
rho(:)=NaN;

for  (k=0:steps)
    [eTot,eVector]=errork(k);
    erd(k+1,1)=eTot;
        for  (j=1:10)
            erd(k+1,1+j)=eVector(j);
        end
        if  (k==0)
                rho(k+1)=NaN;
        else
            rho(k+1)=erd(k+1,col+1)/erd(k,col+1);
```

```
            end
            if(abs(rho(k+1)-0.25)<0.0001)
                break;
    end
end
k=0:steps;
fig1=plot(k,rho(k+1),'k-*');
fprintf('k=         ');fprintf('%11d',k(:));fprintf('\n');
fprintf('error=  ');fprintf('%11.3g',erd(:,1));fprintf('\n');
fprintf('rho=');fprintf('%11.5g',rho(:));fprintf('\n');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [errtot,errvector]  =  errork(k)
numsol=sdflux(diffusolve(k));
anasol=sdflux(asolution(k));
errvector=numsol-anasol;
errtot=totalflux(diffusolve(k))-totalflux(asolution(k));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이제 매트랩 커맨드 윈도우에서 다음과 같이 실행해 보면

```
>>  hw5a2(0)
k=          0          1          2          3          4          5          6          7
error=     0.0781     0.0205    0.00519     0.0013   0.000326  8.14e-005        NaN        NaN
rho=   NaN    0.26248    0.25312    0.25078     0.2502    0.25005        NaN        NaN
>>
```



위 결과에서 k는 n=10*2^k 의 k에 해당하여 k가 1 증가할 때마다 매쉬의 개수가 2배씩 증가한다. error는 argument를 0으로 주었기 때문에 total flux 오차에 대해서만 본 것이다. rho는 현재

step의 오차를 먼저 번 step의 오차로 나눈 값이다. 위와 같이 매쉬를 조밀하게 할수록 error reduction ratio가 점점 줄어들어 0.25에 가까워지는 것을 확인할 수 있으며. k=5에서 rho가 0.25에 0.1%내로 정확해지는 것을 확인할 수 있다.

각 subdomain에서의 오차에 대한 reduction을 확인해 보기 위해서 argument에 1부터 10까지 줄 수 있다.

```
>> hw5a2(1)
k=       0        1        2        3        4        5        6        7
error=  0.0781   0.0205   0.00519  0.0013   0.000326   NaN      NaN      NaN
rho=     NaN    0.25236  0.25059  0.25015  0.25004    NaN      NaN      NaN
>> hw5a2(2)
k=       0        1        2        3        4        5        6        7
error=  0.0781   0.0205   0.00519  0.0013   0.000326   NaN      NaN      NaN
rho=     NaN    0.25179  0.25044  0.25011  0.25003    NaN      NaN      NaN
>> hw5a2(3)
k=       0        1        2        3        4        5        6        7
error=0.0781     0.0205   0.00519   NaN      NaN      NaN      NaN      NaN
rho=    NaN     0.24984  0.24993   NaN      NaN      NaN      NaN      NaN
>> hw5a2(4)
k=       0        1        2        3        4        5        6        7
error=   0.0781   0.0205   0.00519  0.0013   0.000326 8.14e-005   NaN      NaN
rho=      NaN    0.24023  0.24729  0.2493   0.24982  0.24996    NaN      NaN
>> hw5a2(5)
k=       0        1        2        3        4        5        6        7
error=   0.0781   0.0205   0.00519  0.0013   0.000326 8.14e-005 2.04e-005   NaN
rho=      NaN    0.28825  0.25875  0.25214  0.25053  0.25013  0.25003    NaN
>> hw5a2(6)
k=       0        1        2        3        4        5        6        7
error=  0.0781   0.0205   0.00519  0.0013   0.000326 8.14e-005   NaN      NaN
rho=     NaN    0.26596  0.25394  0.25098  0.25025  0.25006    NaN      NaN
>> hw5a2(7)
k=       0        1        2        3        4        5        6        7
error=   0.0781   0.0205   0.00519  0.0013   0.000326 8.14e-005   NaN      NaN
rho=      NaN    0.26305  0.25325  0.25081  0.2502   0.25005    NaN      NaN
>> hw5a2(8)
k=       0        1        2        3        4        5        6        7
error=   0.0781   0.0205   0.00519  0.0013   0.000326 8.14e-005   NaN      NaN
rho=      NaN    0.26217  0.25304  0.25076  0.25019  0.25005    NaN      NaN
>> hw5a2(9)
k=       0        1        2        3        4        5        6        7
error=  0.0781   0.0205   0.00519  0.0013   0.000326 8.14e-005   NaN      NaN
```

rho= NaN 0.26191 0.25299 0.25075 0.25019 0.25005 NaN NaN
>> hw5a2(10)

k=	0	1	2	3	4	5	6	7
error=	0.0781	0.0205	0.00519	0.0013	0.000326	8.14e-005	NaN	NaN
rho=	NaN	0.26193	0.25299	0.25075	0.25019	0.25005	NaN	NaN

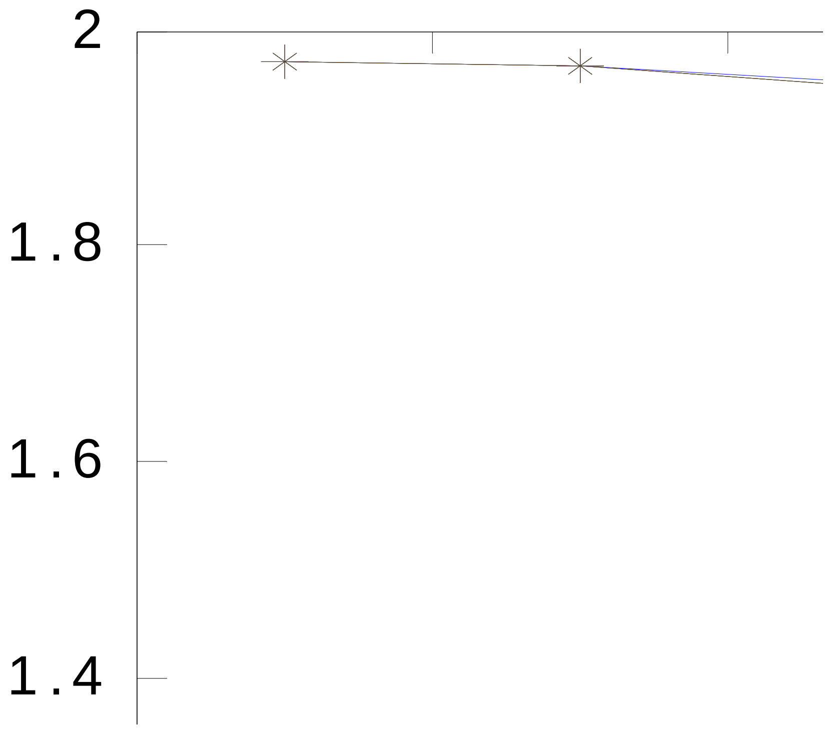
>>

약간 지루하긴 하지만 위의 결과로부터 흥미로운 사실을 알 수 있는데 각 subdomain에서도 오차의 감소율은 0.25로 수렴하며, 수렴하는 속도는 대략적으로 total flux 오차의 수렴속도와 비슷하지만 어떤 domain에서는 더욱 빨리 수렴하는 것을 알 수 있다. 한편 오차 감소율이 0.25보다 작은 상태에서 0.25로 수렴할 수도 있음을 확인 할 수 있다.

이제 flux 그래프를 그려보면

```
function [] = hw5a2gr(steps)
%step에 k의 최대step을 받아서 그래프로 그린다.
y(1,:)=sdflux(asolution(0)); %analytic 함수 flux 저장
z(1,:)=zeros(1,10);
for (k=0:steps)
    y(2+k,:)=sdflux(diffusolve(k)); %k step에서 flux 저장
        [eTot,eVector]=errork(k);
        z(2+k,:)=eVector;%k step에서의 오차 벡터저장
end

x=0.5:9.5; %subdomain의 중심에서의 x값을 저장
plot(x,y,'*-');
%plot(x,z,'*-');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [errtot,errvector] = errork(k)
numsol=sdflux(diffusolve(k));
anasol=sdflux(asolution(k));
errvector=numsol-anasol;
errtot=totalflux(diffusolve(k))-totalflux(asolution(k));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In matlab command window
>> hw5a2gr(5)

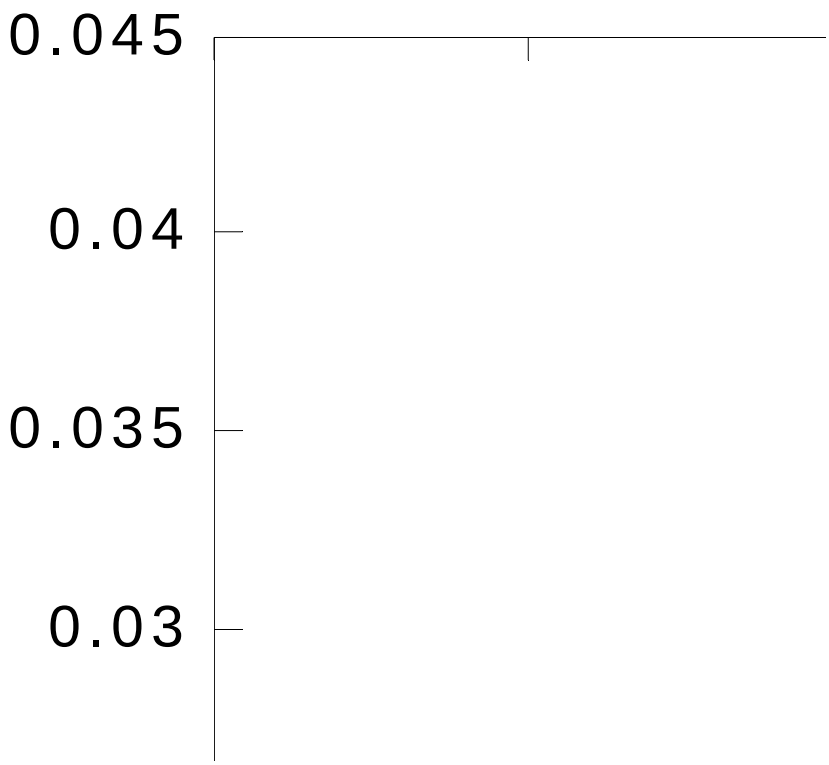


그래프간 간격이 조밀하여 확인하기가 쉽지는 않지만 k step이 증가하여 mesh가 조밀해질수록 해가 analytic solution에 가까워 지는 것을 확인할 수 있다.

각 구간에서의 오차를 각 k step 마다 확인 해보기 위해 위의 matlab 프로그램을 조금만 수정하고 이름을 바꿔서 실행해본다.

```
function [] = hw5a2gr2(steps)

y(1,:)=sdflux(asolution(0)); %analytic 함수 flux 저장
z(1,:)=zeros(1,10);
for (k=0:steps)
    y(2+k,:)=sdflux(diffusolve(k)); %k step에서 flux 저장
          [eTot,eVector]=errork(k);
          z(2+k,:)=eVector;%k step에서의 오차 벡터저장
end

x=0.5:9.5; %subdomain의 중심에서의 x값을 저장
%plot(x,y,'*-');
plot(x,z,'*-');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [errtot,errvector] = errork(k)
```

```
numsol=sdflux(diffusolve(k));
anasol=sdflux(asolution(k));
errvector=numsol-anasol;
errtot=totalflux(diffusolve(k))-totalflux(asolution(k));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
>>hw5a2gr2(5)
```



k가 작을때에는 오차가 a=10 boundary 근처에서 비교적 크고 a=0 boundary 근처에서는 작은데, k가 증가하여 mesh가 조밀해 질수록 오차가 0으로 줄어드는 것을 확인 할 수 있다. 한편 x=4근처에서는 오차가 작게 나오는 것을 확인할 수 있어서 이전에 각 subdomain에 대해 rho가 1/4로 수렴하는 속도가 다르게(domain 4에서 작게 4와 5에서 작게 나온) 나온 이유를 짐작할 수 있다.

B. Non Uniform Source

the analytic solution can not be obtained, hence the true solution needs to be predicted. Now follow the same steps as the above except using previous step solution instead of the analytic solution. What would be the predicted value of the true total flux? Use the extrapolation scheme discussed in the classes.

텐트모양 소스function을 쓰기 위해서는 앞서 작성한 diffusolve.m에서 swc를 2와 같은 1이 아닌 다른 수로 넣어주면 된다. swc=2; 로 바꾼 것을 (원래 swc를 input parameter로 넣어줘도 상관없지만) 여기서는 diffusolve2.m으로 이름만 바꿔서 새 파일로 저장하기로 한다.

In file diffusolve2.m

```
function [phi] = diffusolve2(k)
....
swc=2;
.....
%%%%%%%%%%%%%%%%%%
```

한편 extrapolation을 수행하기 위해 다음과 같은 함수를 작성한다.

```
function [] = hw5bexp(steps)
%step에 k의 최대step을 받아서 그래프로 그린다.
% 마지막 k step과 그 전 것 step으로 참 flux를 예상한다.
rho=1/4;
for (k=0:steps)
    y(1+k,:)=sdflux(diffusolve2(k)); %k step에서 flux 저장
end
y(steps+2,:)=1/(1-rho)*y(steps+1,:)-rho/(1-rho)*y(steps,:);

x=0.5:9.5; %subdomain의 중심에서의 x값을 저장
plot(x,y(steps:steps+2,:),'*-');%steps 에 k번째 flux 저장 steps+1에는 k+1번째 flux
% y의 steps+2번째 row에는 예상하고 있는 참 flux 저장한다.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
>>hw5bexp(1)
```

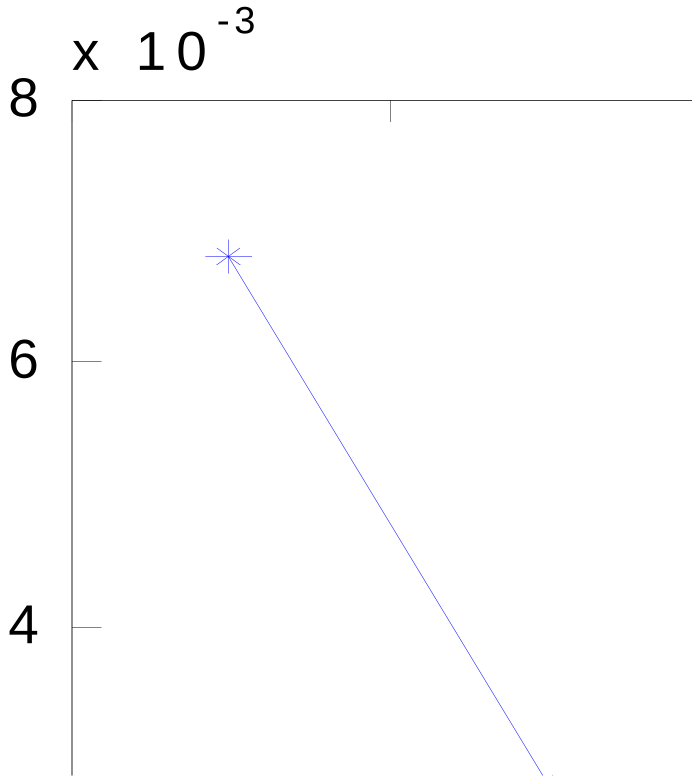
그러면 과연 외삽으로 얻은 flux 가 참 해인지 알아보기 위해 다음과 같이 코드를 수정한다. phi* 추정은 k=0과 k=1일때의 flux를 이용하기로 하고, 이렇게 얻은 flux를 step을 증가시켜 얻은 flux 와의 오차를 plot 하여 k가 증가할수록 오차가 점점 줄어드는지 확인하려고 한다.

function [] = hw5bexp2(steps)

%step에 k의 최대step을 받아서 그래프로 그린다.

```
rho=1/4;

for (k=0:steps)
    y(1+k,:)=sdflux(diffusolve2(k)); %k step에서 flux 저장
end
y(steps+2,:)=1/(1-rho)*y(2,:)-rho/(1-rho)*y(1,:);

x=0.5:9.5; %subdomain의 중심에서의 x값을 저장
plot(x,y(steps:steps+2,:),'*-');

for (k=0:steps)
    err(k+1,:)=y(steps+2,:)-y(k+1,:);
end
```

```
plot(x,err(:,:),'*-');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
>>  hw5bexp2(5)
```



위 그래프는 k=0, 1을 근거로 하여 외삽한 추정해를 기준으로 한 오차를 증가하는 k 에 대해 푼 flux에 대해 plot한 것인데, k가 증가할수록 오차가 줄어드는 것으로 보아 k=0, 1에서 구한 해를 외삽하여 얻은 추정해가 참해와 같거나 매우 비슷하다고 결론 내릴 수 있다.

Part 2. LU-Factorization

1. Write a MATLAB function to perform LU factorization of a given square matrix of Rank n with the following features.(n is arbitrary)

1) Partial Pivoting
2) Produce the permutation matrix (P) as well as L and U such that P=LU

다음과 같은 matlab 함수를 작성한다.
In file lupivot.m

```
function [L,U,P] = lupivot(A)
% 행렬 A 를 받아서 L U factorization을 수행한다.
% pivoting을 문제에 주어진대로 partial pivoting을 사용하며
% scaling은 일단 하지 않기로 한다.
% 프로그래밍을 간단히 하기 위해서 여러가지 부함수를 사용하였으나
% 실제로는 속도를 높이기 위해 subfunction의 argument수를 최소화하거나
% 인라인으로 작성하여 함수 콜에 필요한 overhead를 줄이는 것이 필요하다.
n=rank(A);
P=eye(n);
L=eye(n);

for (k=1:n-1)
    ip=pv(A,n,k);%pivot 찾기
    A=rexch(A,k,ip); %pivotting on Ak
        P=rexch(P,k,ip); %Pk-1 Pk-2...P2 P1
    [L,A]=elimi(L,A,n,k); %L update 및 가우스 소거
end
U=A;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ip] = pv(A,n,k)
%rank n을 같는 행렬 A에 대해 k번째 대각원소에서
%목표로하는 pivot 열 인덱스 찾는 subfunction
ip=k;
mp=abs(A(k,k));
for (i=k+1:n)
    mi=abs(A(i,k));
    if (mi>mp)
        ip=i;
        mp=mi;
```

```
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [A] = rexch(A,i,j)
%행렬 A 와 정수 i, j를 받아서
% A의 i번째 열과 j 번째 열을 바꾸어 리턴한다.
tmpVec=A(i,:);
A(i,:)=A(j,:);
A(j,:)=tmpVec;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [L,A] = elimi(L,A,n,k)
%가우스 소거를 A(k,k)원소에 대해서 수행한다.
%L 행렬을 받아서 multiplier 업데이트하고, 리턴한다.
for (i=k+1:n)
    mik=A(i,k)/A(k,k);
        for (j=k+1:n)
            A(i,j)=A(i,j)-mik*A(k,j);
        end
        A(i,k)=0;%pivot 밑은 지워줌 사실 이 공간에 multiplier를 저장해서
                % L 행렬을 위한 기억장소를 아낄 수 있다.

        L(i,k)=mik; %multiplier를 L의 k열에 추가
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이렇게 만든 함수를 matlab에서 직접 실험해 본다.

```
>> n=3;A=rand(n);
>> [l u p]=lupivot(A)

l =

    1.0000         0         0
    0.1624    1.0000         0
    0.9941    0.4546    1.0000


u =

    0.9706    0.8003    0.9157
```

```
         0    0.3554    0.2731
         0         0   -0.8926


p =

     0     0     1
     0     1     0
     1     0     0

>> [L  U  P]=lu(A)

L =

    1.0000         0         0
    0.1624    1.0000         0
    0.9941    0.4546    1.0000


U =

    0.9706    0.8003    0.9157
         0    0.3554    0.2731
         0         0   -0.8926


P =

     0     0     1
     0     1     0
     1     0     0

>>
>> n=4;B=rand(n);
>> [l  u  p  ]=lupivot(B)

l =

    1.0000         0         0         0
    0.8257    1.0000         0         0
    0.6834    0.0559    1.0000         0
```

```
    0.0372    0.1079    0.9410    1.0000

u =

    0.9595    0.9340    0.3922    0.0318
         0    0.7230    0.1566    0.0450
         0         0    0.4024    0.6749
         0         0         0   -0.3824

p =

     0     1     0     0
     0     0     0     1
     1     0     0     0
     0     0     1     0

>> [L  U  P]=lu(B)

L =

    1.0000         0         0         0
    0.0372    1.0000         0         0
    0.8257    0.1079    1.0000         0
    0.6834    0.0559    0.9410    1.0000

U =

    0.9595    0.9340    0.3922    0.0318
         0    0.7230    0.1566    0.0450
         0         0    0.4024    0.6749
         0         0         0   -0.3824

P =

     0     1     0     0
     0     0     0     1
     1     0     0     0
```

```
     0     0     1     0

>>
```

이러한 것을 보면 matlab의 내장함수 lu()와 같은 결과를 내는 것을 확인할 수 있다.

2. Write a MATLAB program to solve a linear system with LU factors by forward and backward substitution. Verify your program by comparing with the solution obtained with MATLAB's x=A\b function.

다음과 같은 함수를 작성한다.

In file lusolve.m

```
function [x] = lusolve(A,b)
[m,n]=size(b);
if (m<n) %b 벡터를 열벡터로 만든다.
    b=b';
end

n=rank(A);
y=zeros(n,1);

[L,U,P]=lupivot(A);
b=P*b; %PAx=Pb , 즉 LUx=Pb 문제로 바뀐다.

%forward substitution
y(1)=b(1);
for (i=2:n)
    sumod=0.0;
          for (j=1:i-1)
              sumod=sumod+L(i,j)*y(j);
          end
          y(i)=(b(i)-sumod)/1;
end

%backward substitution
x(n)=y(n)/U(n,n);
for (i=n-1:-1:1)
    sumod=0;
          for (j=i+1:n)
              sumod=sumod+U(i,j)*x(j);
          end
          x(i)=(y(i)-sumod)/U(i,i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%
```

다음과 같이 실험해 볼 수 있다.

```
>> A=[1 -1 -1 1;2 -1 1 -1;4 1 -1 1;2 1 -1 -1]

A =

     1    -1    -1     1
     2    -1     1    -1
     4     1    -1     1
     2     1    -1    -1

>> b=[-1 6 0 2]'

b =

    -1
     6
     0
     2

>> x=lusolve(A,b)

x =

     1    -1     1    -2


>> A\b

ans =

     1
    -1
     1
    -2

>>
```

이상의 결과로 보면 x=A\b 로 푼 것과 같은 결과를 내어 L U 분해로 linear system의 직접해를 구하는 프로그램이 잘 작성되었다고 확인할 수 있다.
다른 연립방정식으로 한번 더 확인 해 보면

```
>> A=[4 1 1 2;1 3 0 1;1 0 2 1;2 1 1 4]
```

```
A =

     4     1     1     2
     1     3     0     1
     1     0     2     1
     2     1     1     4

>> b=[8  0  1  6]'

b =

     8
     0
     1
     6

>> x=lusolve(A,b)

x =

    2.0000   -1.0000   -1.0000    1.0000

>> x=A\b

x =

    2.0000
   -1.0000
   -1.0000
    1.0000

>>
```

여전히 잘 맞는 것을 확인 할 수 있다.

<부록> function list

diffusolve.m (1차원 diffusion equation 풀어 flux 구해주는 함수)
sdflux.m (정해진 (10개)subdomain 으로 만들어주기)
totalflux.m (total flux 구하기)
asolution.m (analytic solution을 diffusolve에서 나오는 flux 형태로 보기)
hw5a2.m (오차 감소율이 1/4에 0.01%내로 올때까지의 k를 보여주는 루틴)
hw5a2gr.m (flux를 한 그래프에 plot)
hw5a2gr2.m (오차를 한 그래프에 plot)

diffusolve2.m (source를 텐트모양 함수로 주었을 때)
hw5bexp.m (k와 k+1에 대해서 외삽하여 해를 추정)
hw5bexp2.m (k=0, 1에 대해 외삽하여 추정한 해가 정해임을 오차감소하는 것으로부터 확인)
hw5bgr.m (그냥 k까지 plot해보는 루틴)

lupivot.m (l u p 분해 해주는 루틴)
lusolve.m (lu 분해해서 선형 시스템 풀어주는 루틴)