2008년 2학기 HW#6

수치해석기초

HW#6 : **2-D Particle Diffusion**

원자핵공학과
2003-12491
이 원 재

A. Discretization

The meshes are ordered in the natural way, namely, from the upper left corner to the lower right corner. Write a MATLAB program to discretize the 2-D problem and construct the linear system which consists of 100 equations. For sparse storage, use the function sparse in MATLAB. Use a sparse matrix storage scheme if you use other language.

다음과 같이 일단 input argument로 매쉬사이즈를 받는 없이 linear system을 만드는 matlab 함수를 작성한다.

In lsc2d.m

```
function [A,bs] = lsc2d(meshsize)
%Linear System Constructor 2-D
%meshsize로 x, y축 방향에 공통으로 적용되는 mesh 개수를 받는다
a=5;b=5;
D=2.5;
aph=0.15; %알파값
s=1000;

m=meshsize; %x축 mesh 개수
n=meshsize; %y축 mesh 개수

N=m*n; %총 mesh 개수
hx=a/m; %x축 mesh 크기, 확장성을 위해 변수 구분
hy=b/n; %y축 mesh 크기
hx2=hx*hx;
hy2=hy*hy;
%h=0.5;

B2=aph/D; %B^2 = Alpha/D , Buckling
stilda=s/D; %source term devided by diffusivity

A=sparse(N,N); %sparse matrix declaring
bs=zeros(1,N); %source vector b_source

for (k=1:N)
    if (k>m) % 첫번째 줄, 즉 y=0 인 row 만 지나가면 반드시 위쪽에 셀이 존재한다.
        A(k,k-m)=-1/hy2; %위쪽 셀
    end

     if (mod(k,m)==1)
        ; % 도메인의 왼쪽 끝에 오면 왼쪽 셀에 접근할 수 없다.
    else %그렇지 않다면 접근할 수 있다.
```

```
      A(k,k-1)=-1/hx2; %왼쪽 셀
end

 if (mod(k,m)==0)
   ; % 도메인의 오른쪽 끝에 오면 오른쪽 셀에 접근할 수 없다.
 else %그렇지 않으면 접근할 수 있다.
     A(k,k+1)=-1/hx2; %오른쪽 셀
end

 if (k<=N-m) % 맨 아래쪽 라인(y=b)이 아닌 이상 반드시 아래쪽 셀이 존재한다.
     A(k,k+m)=-1/hy2; %아래쪽 셀
end
 % 중앙 셀
 if (k<=m) %y=0 인 경우 reflective for y
     if (k==1)
       A(k,k)=B2+1/hx2+1/hy2;     %x=0 reflective
      elseif (k==m)
          A(k,k)=B2+3/hx2+1/hy2;     %x=a null flux
      else
          A(k,k)=B2+2/hx2+1/hy2;         % 0<x<a
      end
 elseif (k<=(n-1)*m) % 0<y<b 인 경우
     switch ( mod(k,m) )
         case (1)
             A(k,k)=B2+1/hx2+2/hy2;     %x=0 reflective
         case (0)
             A(k,k)=B2+3/hx2+2/hy2;     %x=a null flux
           otherwise
               A(k,k)=B2+2/hx2+2/hy2;     % 0<x<a
     end
 else %y=b 인경우 nullflux for y
     if (k==(n-1)*m+1)
         A(k,k)=B2+1/hx2+3/hy2;     %x=0 reflective
     elseif (k==N)
         A(k,k)=B2+3/hx2+3/hy2;     %x=a null flux
     else
         A(k,k)=B2+2/hx2+3/hy2;     % 0<x<a
     end
 end

 bs(k)=stilda; % filling source vector b
```

end

%%

이 함수는 입력 파라매터로 매쉬 사이즈를 받아서 A 행렬과 소스텀에 해당하는 벡터를 각각 리턴한다.

```
>> [A,b]=lsc2d(10);
>> size(A)
ans =
    100    100
>> size(b)
ans =
      1    100
>>
```

다음과 같이 100x100 계수행렬과 길이가 100인 소스벡터가 형성된 것을 볼 수 있다. 올바르게 block tridagonal 행렬이 만들어졌는지 확인하기 위해 mesh 사이즈를 줄이고 sparse matrix를 펼쳐서 다음과 같이 확인 해 볼 수 있다.

```
>> full(lsc2d(3))
ans =
```

0.7800	-0.3600	0	-0.3600	0	0	0	0	0
-0.3600	1.1400	-0.3600	0	-0.3600	0	0	0	0
0	-0.3600	1.5000	0	0	-0.3600	0	0	0
-0.3600	0	0	1.1400	-0.3600	0	-0.3600	0	0
0	-0.3600	0	-0.3600	1.5000	-0.3600	0	-0.3600	0
0	0	-0.3600	0	-0.3600	1.8600	0	0	-0.3600
0	0	0	-0.3600	0	0	1.5000	-0.3600	0
0	0	0	0	-0.3600	0	-0.3600	1.8600	-0.3600
0	0	0	0	0	-0.3600	0	-0.3600	2.2200

B. Reference Solution

Use you LU factor function and solver to generate the reference solution. Plot the radial particle distribution using surf. You need to converted the solution vector to a matrix (2-D) for the plot.

위에서 lsc2d.m으로 작성한 system을 풀기 위해 앞선 숙제#5에서 작성한 lupivot.m과 lusolve.m을 수정한 다음과 같은 함수를 작성한다.

In file lusol2d.m

```
function [x,cnt] = lusol2d(A,b)
%LU factorization solver for 2D diffusion equation
%This function returns exact solution vector which can be used for ref. sol
[m,n]=size(b);
if (m<n) %b 벡터를 열벡터로 만든다.
    b=b';
end

%
cnt=0;%for FLOPs count
%

[n,n]=size(A);
y=zeros(n,1);
%%%%
%%
[n,n]=size(A);

P=eye(n);
L=eye(n);

for (k=1:n-1)
    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%pivot 찾기
    %row exchange
```

```
      tmpVec=A(k,:);
      A(k,:)=A(ip,:);
      A(ip,:)=tmpVec;

    tmpVec=P(k,:);
      P(k,:)=P(ip,:);
      P(ip,:)=tmpVec;


      %L update 및 가우스 소거
      for (i=k+1:n)
            mik=A(i,k)/A(k,k); cnt=cnt+1;
            for (j=k+1:n)
                  A(i,j)=A(i,j)-mik*A(k,j); cnt=cnt+1;
            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
end
U=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); cnt=cnt+1;
      end
      y(i)=(b(i)-sumod)/1; cnt=cnt+1;
end

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

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

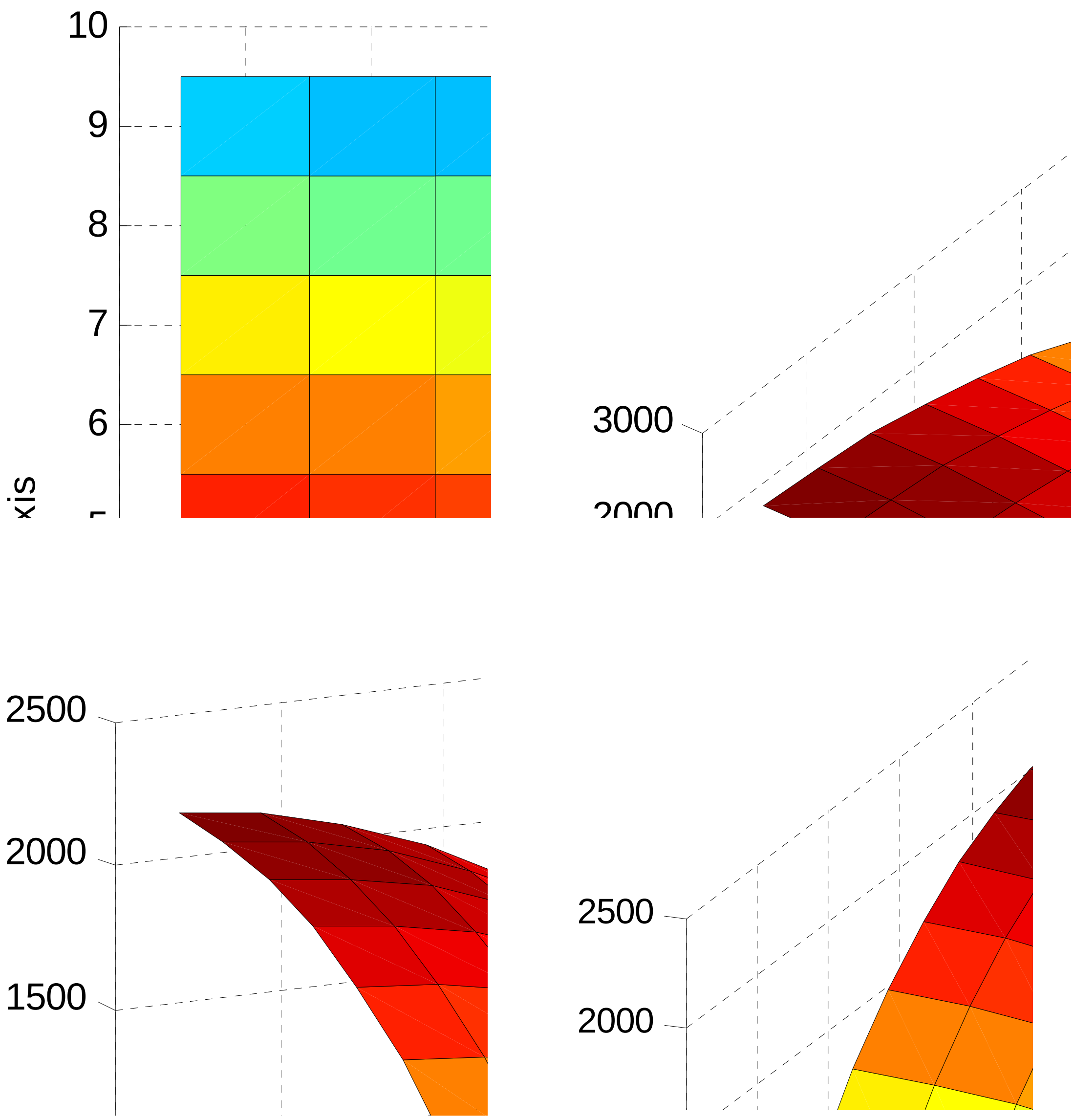
본래 lusolve 가 A 행렬과 b 벡터를 받아서 lupivot function call을 통해 LU 분해를 수행하였고, lupivot 파일에도 여러 부함수가 포함되어 있었지만 부함수 call 시간을 줄이고 FLOPs 계산에 필요한 cnt 변수를 공유하기 위해 모두 한 개의 함수에 합하였다. 시스템 풀이에 필요한, A와 b를 넘겨받은후에 LU 분해를 수행한뒤 substitution을 두 번 시행하여 얻은 해벡터 x와, FLOPs count 를 리턴한다. 해벡터 x 는 natural numbering이 된 domain의 해를 의미한다.

```
>>x=lusol2d(A,b);
```

이렇게 얻은 해를 plot 하기 위해 다음과 같이 그래프 그리는 함수를 작성한다.

```
function [phi,xx,yy] = getgraph(x)
%%%%%%%%%%%%%%%%%%%
a=10;b=10; %dimension size in cm
[r,c]=size(x);
%m=10;n=10; %adjustment needed when we want change grid size!!
%%%%%%%%%%%%5
n=sqrt(r*c); %간단을 위해 x축 격자와 y 축 격자가 같다고 가정한것
m=n;
xx=zeros(1,m);
yy=zeros(1,n);

phi=zeros(m,n);
for (i=1:n)
    phi(:,i)=x(((i-1)*n+1):(i*m)); %x 축 과 y 축의 격자수가 서로 같은것을 가정
end

for (k=1:m)
    xx(k)=a/m*(k-1)+a/(2*m); %x -> 0.5 1.5 2.5 ... 9.5
end
for (k=1:n)
    yy(k)=b/n*(k-1)+b/(2*n);
end

surf(xx,yy,phi);
xlabel('X Axis')
```

ylabel('Y Axis')
%%
해 벡터 x를 인자로 받아서 phi의 행렬(m by n)을 작성하고 격자를 나눠서 3차원 그래프를 그려주고 phi 와 x, y 격자 (중간)점을 리턴한다.
>>phi=getgraph(x);



C. Gauss-Seidel Iterative Solution

Write a program to implement the Gauss-Seidel method. Perform the iteration and estimate the spectral radius of the iteration matrix by computing the ratio of pseudo error norms at each iteration step.

$$\rho = \frac{\left\| x^{(k)} - x^{(k-1)} \right\|}{\left\| x^{(k-1)} - x^{(k-2)} \right\|}$$

Compare this with the theoretical spectral radius which can be obtained by computing the maximum eigenvalue of the spectral radius of the iteration matrix. Exit the iteration when the relative change of the pseudo error becomes less than 10-6. The relative pseudo error is defined as:

$$\tilde{\rho} = \frac{\left\| x^{(k)} - x^{(k-1)} \right\|}{\left\| x^{(k)} \right\|}$$

다음과 같은 매트랩 함수를 작성한다. 이 함수는 두 개의 subfunction을 갖고 있다.
In file gsisol2d.m

```
function [xk,graphmat,rho_est] = gsisol2d(A,b)
%Gauss-Seidel Iteration SOLver for 2D diffusion equation
%첫번째 반환인자는 최종적으로 얻은 해벡터, 두번째 반환인자는
%k-th iteration에 대해 오차, 상대오차, spectral radius를 저장하고 있는 행렬

[n,n]=size(A); %system 의 사이즈를 취함,
%%%%%%%%%%%%%%%%%%%%%%%%constants%%%%%%%
maxIter=500; %최대 반복횟수
epsilon=1.e-6;
%%%%해 벡터 초기화%%%%%%%%%%%%%%%%%%%%%%%
xk=ones(1,n); %current solution vector x^k
xk_1=ones(1,n); %previous solution vector ; x^(k-1)
% 초기 추측해는 원소가 모두 1인 것으로 함
%%%% 오차 관련 벡터들 초기화%%%%%%%%%
errk_1=zeros(1,n);
errk=zeros(1,n);
errk_1(:)=NaN;
errk(:)=NaN;
rho_est=NaN;
%%%% 그래프 그리기위해 준비한 행렬
graphmat=zeros(1,4);% k, ||err||, relative ||err|| , rho
```

```
for (k=1:maxIter)
    errk_1=errk; %먼저번 step에서의 오차 벡터를 예전 오차벡터 저장용에 복사
     for (i=1:n)
        suml=0.0;
           sumu=0.0;
           for (j=1:i-1)
                suml=suml+A(i,j)*xk(j);
        end
           for (j=i+1:n)
               sumu=sumu+A(i,j)*xk_1(j);
           end
           xk(i)=(b(i)-suml-sumu)/A(i,i);
    end

     errk=xk-xk_1; %새로운 오차벡터 취함
    if(k>=2)
         rho_est=norm2(errk)/norm2(errk_1); %spectral radius 추측
     end
    rho_tilda=norm2(errk)/norm2(xk); %오차의 상대값

     graphmat(k,1)=k;
     graphmat(k,2)=norm2(errk);
     graphmat(k,3)=rho_tilda;
     graphmat(k,4)=rho_est;

    if( rho_tilda< epsilon)
         break;
     end
     xk_1=xk; %예전것에 현재 나온 것을 대입
end
prn_info(graphmat);

%%%%%%%%%%%%%%%%%%%%
function [graphmat] = prn_info(graphmat)
%%%%%fprintf('k , ||error|| , rho =%8d ,%8g ,%8g \n',k,norm2(errk),rho_est);

[AX,H1,H2] = plotyy(graphmat(:,1),graphmat(:,2),graphmat(:,1),graphmat(:,4));%k값에 따른 오차벡터의
크기 plot
xlabel('k')
set(get(AX(1),'Ylabel'),'string','||error vector||')
set(get(AX(2),'Ylabel'),'string','rho')
```

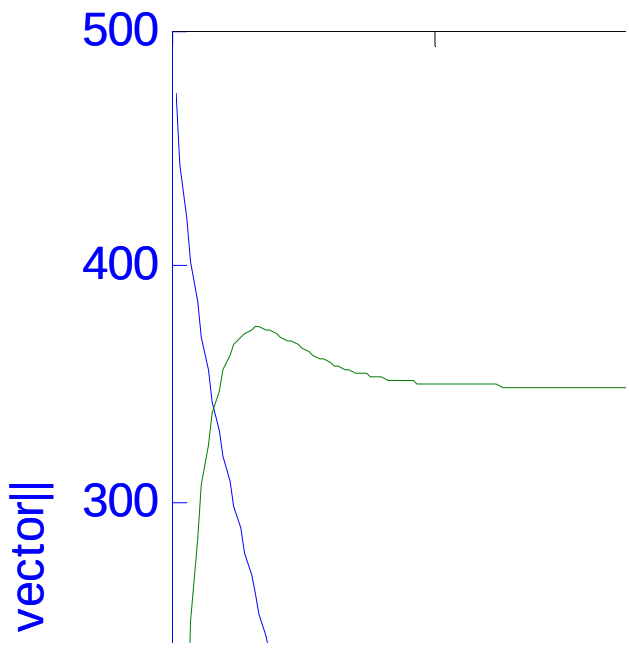
```
title('k vs error, rho')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [r] = norm2(errvec)
[m,n]=size(errvec);
n=m*n;
sum2=0.0;
for (k=1:n)
    sum2=sum2+errvec(k)*errvec(k);
end
r=sqrt(sum2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이 함수를 다음과 같이 실행 시키면

```
>> [xgsi,knes,rho]=gsisol2d(A,b);
>>format short g
>> knes
kner =

      1       473.88     0.97946        NaN
      2       443.38     0.47967    0.93564
      3       420.86     0.31373    0.94922
      4        401.9     0.23109    0.95495
      5        385.2     0.18172    0.95843
      6        370.1     0.14893     0.9608
      7       356.22      0.1256    0.96251
     ...       ....        ....           ...
    290     0.015271  1.1987e-006   0.96492
    291     0.014735  1.1566e-006   0.96492
    292     0.014218  1.1161e-006   0.96492
    293      0.01372  1.0769e-006   0.96492
    294     0.013238  1.0391e-006   0.96492
    295     0.012774  1.0027e-006   0.96492
    296     0.012326  9.6751e-007   0.96492

>>
```

첫 번째 열은 k 값이고, 두 번째 열은 그 행의 k 값에 대응하는 error norm이다. 세 번째 열은 relative pseudo error 이고, 마지막 네 번째 열은 spectral radius 이다. 마지막 시행에서 k는 296이고, 이때에 목표로 했던 1.e-6보다 적은 상대오차가 얻어짐을 알 수 있다. 즉 1.e-6 상대오차를 얻기 위해서는 반복법으로 296회의 시행이 필요함을 알 수 있다.

함수 실행시 자동으로 다음과 같은 그래프가 그려진다.



아래쪽에 exponential 비슷하게 감소하는 그래프는 오차벡터의 norm 이고, 위쪽에 그려진 그래프는 spectral radius를 각 k에 대해서 오차벡터의 norm의 비율을 통해서 추정한 값이다. 오차는 반복 횟수 k 가 증가함에 따라 급격히 감소하면서 296번째 시행에서는 0.012326 이 되는데, 이는 초기오차 473.88 에 spectral radius 0.96492를 296회 곱한 값, 즉

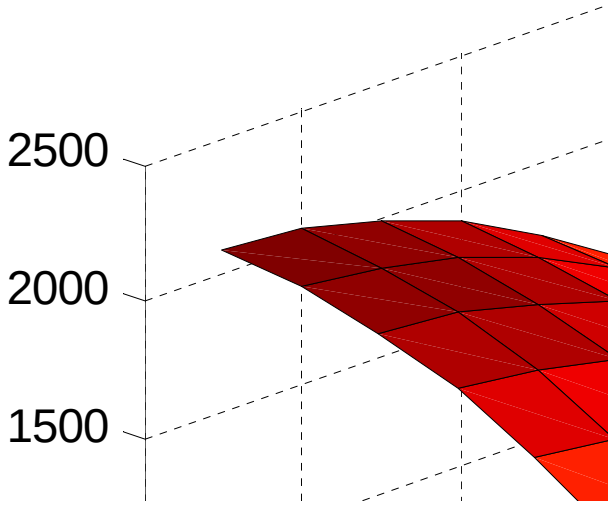
```
>> 473.88*0.96492^296
ans =
       0.012165
```

와 비슷하다는 것을 확인할 수 있다.

여기서 한 가지 흥미로운 사실은 k가 증가하면서 초기에는 spectral radius의 추정치가 작았던 것이 스무번정도 반복하면서 점점 증가했다가 다시 감소하여 0.96492 에 수렴한다는 사실이다. 이는 iteration 행렬 T의 고유값들에 의해 발생하는 패턴으로 해석할 수 있다.

한편 Gauss-Seidel 반복법으로 얻은 해를 plot 해보면

```
>>phigsi=getgraph(xgsi);
```



직접해로 얻은 것과 같거나 매우 비슷한 결과를 내고 있음을 알 수 있다.

이제 G-S iteration으로 얻은 spectral radius와 T=-inv(M)*N의 최대 eigenvalue 가 과연 일치하는지 확인하기 위해 다음과 같은 함수를 작성한다. eigenvalue를 구하는 루틴은 matlab 내장 함수를 사용하였다.

In file eigtgsi.m

```
function [g] = eigtgsi(A)
%EIGen value for T GSI method matrix
%%%%%%%%%%theorytical spectral radius
%반환인자는  Gauss-Seidal iteration 행렬의
%고유치를 크기순서대로 6개 갖고 있는 벡터이다. (matlab 내부함수 사용)
[n,n]=size(A);
M=sparse(n,n);  %M=L+D
N=sparse(n,n);  %N=U

for  (k=1:n)
    for  (j=1:k)
        M(k,j)=A(k,j);
    end
     for  (j=k+1:n)
         N(k,j)=A(k,j);
     end
end
```

```
T=-inv(M)*N;
g=eigs(T);
%%%%%%%%%%%%%%%%%%%%%%%%%
>> g=eigtgsi(A)
g =
        0.96492
        0.86414
        0.86414
        0.76855
        0.69169
        0.69169
>>
```

A 로부터 얻은 T 행렬의 고유치 6개를 크기 순서대로 리턴한 것이다. 가장 큰 고유치 0.96492 가 spectral radius 의 추정값 0.96492 와 일치하는 것을 확인할 수 있다.

D. SOR Iterative Solution

Change the Gauss-Seidel code to a SOR code which implement elementwise extrapolation. Increase the over-relaxation parameter (w) from 1 with a step size of 0.1 until observe an increase in the estimated spectral radius. If there is a turn-over of the estimated spectral radius, step back one step and refine the step size by 1/10. Then try to increase w again to determine the optimum w which gives the minimum spectral radius. Compare the number of iterations needed to achieve the same error for the SOR and G-S cases. You can obtain the exact error of an iterate by comparing with the reference solution. Compare the estimated optimum w with that can be determined analytically by using Young's formula.

다음과 같은 함수를 작성한다.
In file sorsol2d.m

```
function [opt_xk,wr,opt_w,min_rho] = sorsol2d(A,b)
%Successive Over-Relaxation Method modifying GSI SOLver for 2D diffusion equation

%첫번째 반환인자는 최종적으로 얻은 해벡터, 두번째 반환인자는
% weightig factor와 spectal radius를 저장한 행렬
%세번째 네번째 반환인자는 각각 최적 w와 최소(최적) spectral radius

rho_e0=0; %spectral radius for current
rho_e1=1; %spcctral radius for previous

w=1.0; %weighting factor
%%%% 그래프 관련 행렬

wr=zeros(1,2);

incr=0.1; %0.1씩 w를 증가시켜감
for (w=1.0:incr:2)
    rho_e0=gsi(w,A,b);
     if (rho_e0>rho_e1) %넘어서면 한발짝 물러서고 loop 종료
        w=w-incr;
           break;
     else
        rho_e1=rho_e0;
           continue;
    end
end
incr=0.01;
cnt=1;
```

```
min_rho=1;
opt_w=1.0;
for (w=w:incr:w+0.1) %현재 w에는 앞서 저장된 값이 남아있음
    [rho,xk]=gsi(w,A,b);
    fprintf('w , rho = %8g , %8g \n',w,rho);
    wr(cnt,1)=w;
    wr(cnt,2)=rho;
    cnt=cnt+1;
    if( rho<min_rho )
        min_rho=rho;
        opt_xk=xk;
        opt_w=w;
    end
end
fprintf('optimum w = %g \n',opt_w);
fprintf('spectral radius = %g \n',min_rho);

%%%%%%%%%%%%%%%%%%%%%%%%%%
function [rho_0,xk] = gsi(w,A,b)
[n,n]=size(A);
maxIter=500; %최대 반복횟수
epsilon=1.e-5;
xk=ones(1,n); %current solution vector x^k
xk_1=ones(1,n); %previous solution vector ; x^(k-1)
% 초기 추측해는 원소가 모두 1인 것으로 함
%%%% 오차 관련 벡터들 초기화%%%%%%%%%
errk_1=zeros(1,n);
errk=zeros(1,n);
errk_1(:)=NaN;
errk(:)=NaN;

rho_0=0;
rho_1=1;

for (k=1:maxIter)
   errk_1=errk; %먼저번 step에서의 오차 벡터를 예전 오차벡터 저장용에 복사
    for (i=1:n)
       suml=0.0;
        sumu=0.0;
        for (j=1:i-1)
            suml=suml+A(i,j)*xk(j);
```

```
            end
              for  (j=i+1:n)
                   sumu=sumu+A(i,j)*xk_1(j);
              end
              xk(i)=w*(b(i)-suml-sumu)/A(i,i)+(1-w)*xk_1(i);
     end

      errk=xk-xk_1; %새로운 오차벡터 취함
     if(k>=2)
          rho_0=norm2(errk)/norm2(errk_1); %spectral radius 추측
      end
%     rho_tilda=norm2(errk)/norm2(xk); %오차의 상대값

     if( abs(rho_0 - rho_1)<epsilon) % spectral radius가 수렴하면 iteration 종료
          break;
      end
      xk_1=xk; %예전것에 현재 나온 것을 대입
      rho_1=rho_0; %예전 spectral radius에 현재 spectral radius를 대입
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [r] = norm2(errvec)
[m,n]=size(errvec);
n=m*n;
sum2=0.0;
for (k=1:n)
     sum2=sum2+errvec(k)*errvec(k);
end
r=sqrt(sum2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

첫 번째 반환인자는 최종 해벡터이고, 두 번째 반환인자는 0.01단위로 weighting factor와 spectral radius를 저장한 행렬이다.

```
>> [xsor,wr,wo,ro] = sorsol2d(A,b);
w , rho =      1.6   , 0.843026
w , rho =     1.61   , 0.836162
w , rho =     1.62   , 0.828273
w , rho =     1.63   , 0.819309
w , rho =     1.64   , 0.808875
w , rho =     1.65   , 0.796677
w , rho =     1.66   , 0.781728
```

```
w , rho =      1.67   , 0.761951
w , rho =      1.68   , 0.730064
w , rho =      1.69   , 0.852872
w , rho =       1.7   ,  1.41067
optimum w = 1.68
spectral radius = 0.730064
>>
```

이와 같이 실행하면 최적 weighting factor는 1.68이고 이때의 spectral radius는 약 0.73임을 알 수 있다.

이제 iteration 회수를 비교하기 위해 다음과 같이 함수를 수정한다.

In file wgsiter.m

```
function [erriter,itererr,xk] = wgsiter(w,A,b,xref)
%첫번째 반환인자에는 해당 오차에 대한 iteration 횟수
%두번째 반환인자에는 해당 iteration 횟수에 대한 오차
%마지막 반환인자에는 해벡터가 들어간다.
[n,n]=size(A);
maxIter=500; %최대 반복횟수
epsilon=1.e-6;
xk=ones(1,n); %current solution vector x^k
xk_1=ones(1,n); %previou solution vector x^k-1
% 초기 추측해는 원소가 모두 1인 것으로 함
%%%% 오차 관련 벡터들 초기화%%%%%%%%%%
errk_1=zeros(1,n);
errk=zeros(1,n);
errk_1(:)=NaN;
errk(:)=NaN;

rho_0=0;
rho_1=1;

k=1;m=1;l=1;
eseq=[2 ,1 , 0.5 , 0.2, 0.1, 0.05, 0.01]; %임의의 오차 sequence, 상대오차
[len_r,len_c]=size(eseq);
n_eseq=len_r*len_c;
erriter=zeros(1,2);
itererr=zeros(1,2);

    for (k=1:maxIter)
        errk_1=errk; %먼저번 step에서의 오차 벡터를 예전 오차벡터 저장용에 복사
        for (i=1:n)
```

```
            suml=0.0;
            sumu=0.0;
            for  (j=1:i-1)
                 suml=suml+A(i,j)*xk(j);
            end
            for  (j=i+1:n)
                 sumu=sumu+A(i,j)*xk_1(j);
            end
            xk(i)=w*(b(i)-suml-sumu)/A(i,i)+(1-w)*xk_1(i);
        end

        errk=xk-xref; %새로운 오차벡터 취함 % x ref 에 대한 정확한 오차
%    if(k>=2)
%       rho_0=norm2(errk)/norm2(errk_1); %spectral radius 추측
%    end
        rho_tilda=norm2(errk)/norm2(xk); %오차의 상대값 exact relative error


        if(mod(k,10)==0)
            itererr(l,1)=k;
            itererr(l,2)=rho_tilda;
            l=l+1;
        end

        if( rho_tilda< epsilon)
           break;
        end

        if( abs(rho_tilda-eseq(m))<eseq(m)/10) %오차의 상대값이 목표한 오차와 10%이내에 맞으면
현재 iteration count 저장하고
                                              %다음번 오차 시퀀스로 감
            erriter(m,1)=rho_tilda;
            erriter(m,2)=k;
            m=m+1;
        end
        if(m>n_eseq)
            break;
      end
        xk_1=xk; %예전것에 현재 나온 것을 대입
%    rho_1=rho_0; %예전 spectral radius에 현재 spectral radius를 대입
    end
```

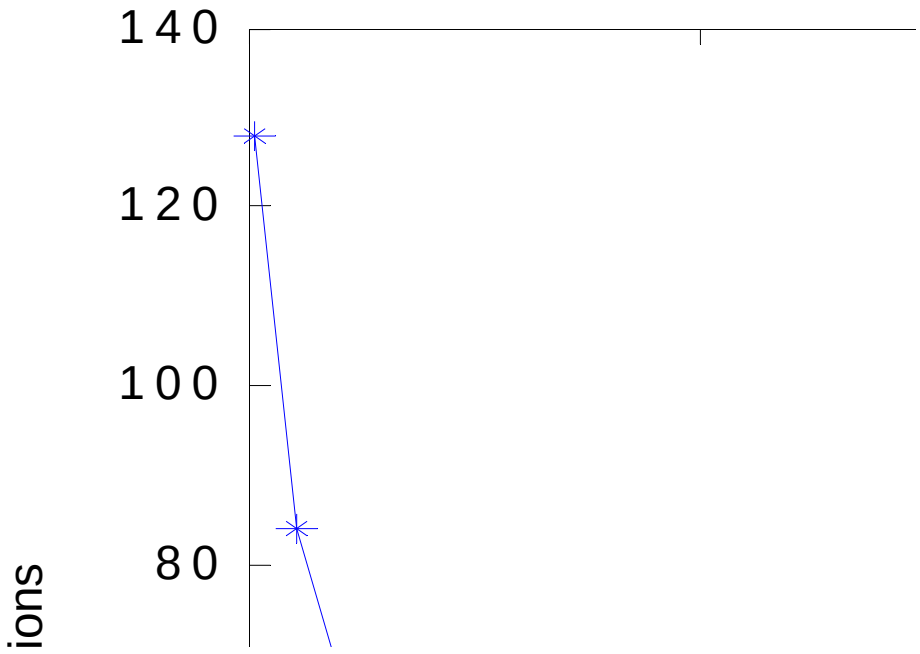
```
function [r] = norm2(errvec)
[m,n]=size(errvec);
n=m*n;
sum2=0.0;
for (k=1:n)
    sum2=sum2+errvec(k)*errvec(k);
end
r=sqrt(sum2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이 함수는 4개의 input parameter를 weighting factor, A, b, exact solution vector 의 순서로 받는다. 여기서의 오차는 exact solution vector 와의 차로 얻은 것으로서 이것의 상대오차를 취한후 임의로 결정한 상대오차값들에 근접하면 그때의 반복횟수를 저장한다. 반대로 10단위의 iteration 마다 오차를 저장하며, 반복법의 loop는 상대오차가 epsilon=1.e-6 값보다 작아지면 종료된다.
이 함수를 이용하기 위해서 다음과 같은 도움함수를 작성하면

```
function [] = cmpgsisor(w)
% CoMPare GSI and SOR of W
[A,b]=lsc2d(10);
xref=lusol2d(A,b);
[eIter1,iterE1,x1]=wgsiter(1,A,b,xref);%보통 GS 반복법
[eIter2,iterE2,x2]=wgsiter(w,A,b,xref);% w weigting SOR

[r1,c1]=size(iterE1);
[r2,c2]=size(iterE2);

fprintf('GSI\n');
for (k=1:r1)
    fprintf('iteration , error = %g  , %g \n',iterE1(k,1),iterE1(k,2));
end
[r2,c2]=size(iterE2);

fprintf('SOR\n');
for (k=1:r2)
    fprintf('iteration , error = %g  , %g \n',iterE2(k,1),iterE2(k,2));
end

plot(eIter1(:,1),eIter1(:,2),'b*-',eIter2(:,1),eIter2(:,2),'bo-');
xlabel('||error||/||x(k)||')
ylabel('Iterations')
legend('Gauss-Seidel','SOR',2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

이제 matlab command window에서 다음과 같이 입력하면

```
>> cmpgsisor(1.68);
GSI
iteration , error = 10   , 2.37756
iteration , error = 20   , 0.997392
iteration , error = 30   , 0.543195
iteration , error = 40   , 0.328319
iteration , error = 50   , 0.209562
iteration , error = 60   , 0.138105
iteration , error = 70   , 0.0928449
iteration , error = 80   , 0.0632278
iteration , error = 90   , 0.0434285
iteration , error = 100   , 0.0300019
iteration , error = 110   , 0.020808
iteration , error = 120   , 0.0144707
SOR
iteration , error = 10   , 0.216989
iteration , error = 20   , 0.00994633
iteration , error = 30   , 0.000428797
iteration , error = 40   , 1.8548e-005
iteration , error = 50   , 7.92511e-007
>>
```



```
>> wgsiter(1,A,b,x)

ans =

    2.1282    11
    1.0702    19
   0.54319    30
   0.21881    49
   0.10862    66
  0.054361    84
  0.010837   128

>> wgsiter(1.68,A,b,x)

ans =

    1.8703     3
   0.92869     5
   0.51805     7
   0.21699    10

>>
```

이와 같은 결과를 얻는다. 그래프를 보면 비슷한 오차수준 예를 들어 0.5에서 SOR 법은 7회, G-S 법은 30회가 필요한 것을 알 수 있다.

한편 w를 바꿔가며 반복적으로 SOR을 사용해서 얻어낸 optimum w가 이론적인 식으로 얻은 값과는 얼마나 차이가 나는지를 확인 해 보면

```
>> rho_gs=0.96492

rho_gs =

    0.96492

>> w_opt=2*(1-sqrt(1-rho_gs))/rho_gs

w_opt =

    1.6845
>>
```

실험적으로 찾아낸 w=1.68과 비슷함을 알 수 있다.

E. Effect of Mesh Size

Investigate the effect of the mesh size on the convergence of the iteration scheme and also on the memory and computation requirement for the direct method. To begin with use the mesh size of 1cm. Now you may not be able to obtain the maximum eigenvalue of the iteration matrix from MATLAB. Further reduce the mesh size and see what happens to the estimated spectral radius of the Gauss-Seidel iteration scheme.

메쉬 사이즈를 증가시켜 갈 때 수렴성의 변화를 보기 위해서 다음과 같은 함수를 작성한다.

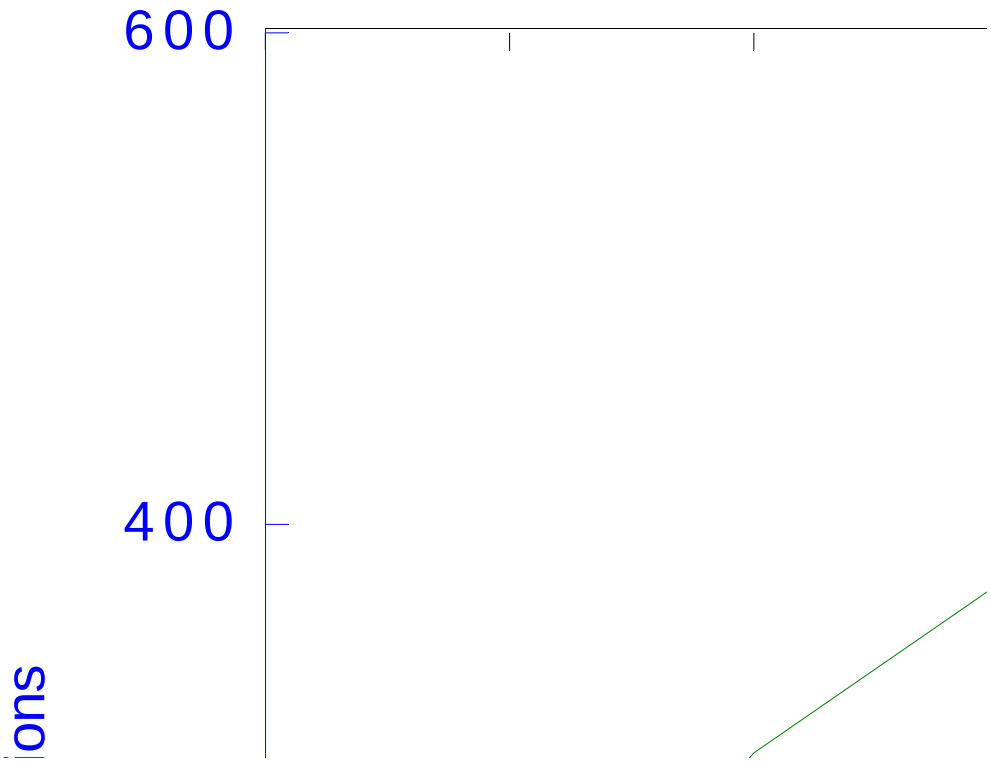
```
function [grph] = vmsgsi(sizebound)
%Varing Mesh Size for GSI
grph=zeros(1,3);

i=1;
for (m=5:sizebound)
    [A,b]=lsc2d(m);
     [xk,knes,rho]=gsisol2d(A,b);
     [r,c]=size(knes);
     grph(i,1)=m;
     grph(i,2)=knes(r,1); %iteration 횟수
     grph(i,3)=knes(r,4); %spectral radius
     i=i+1;
end

[AX,H1,H2] = plotyy(grph(:,1),grph(:,2),grph(:,1),grph(:,3));%k값에 따른 오차벡터의 크기 plot
xlabel('mesh')
set(get(AX(1),'Ylabel'),'string','iterations')
set(get(AX(2),'Ylabel'),'string','spectral radius')
title('mesh size vs error, rho')
```

bound를 15까지 주면 컴퓨터에 따라서는 오랜 시간이 걸리기도 한다.

```
>> vmsgsi(15)
ans =
     5    77    0.85257
     6   110    0.89878
     7   149    0.92648
     8   193    0.94429
     9   242    0.95637
    10   296    0.96492
    11   354     0.9712
```

12	417	0.97594
13	484	0.9796
14	500	0.98249
15	500	0.9848

>>

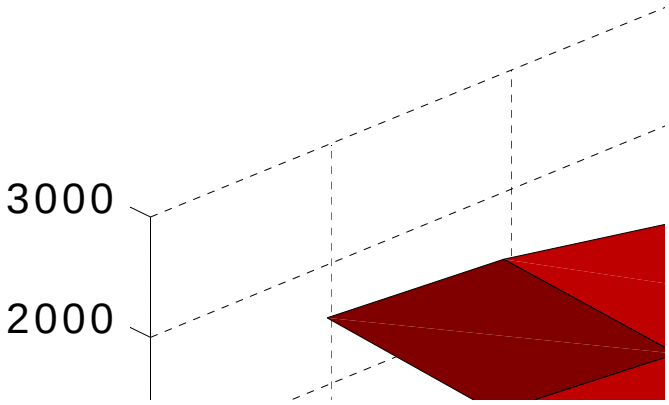


mesh size를 증가시켜 가면 원하는 오차수준의 해를 얻기 위해 필요한 iteration 수가 증가하는 것을 확인 할 수 있다. mesh size를 13으로 했을 때 484 번의 반복이 필요했고 그 추세를 볼 때 14 mesh size일 때에는 600번에 가까운 반복이 필요한 것으로 생각된다.

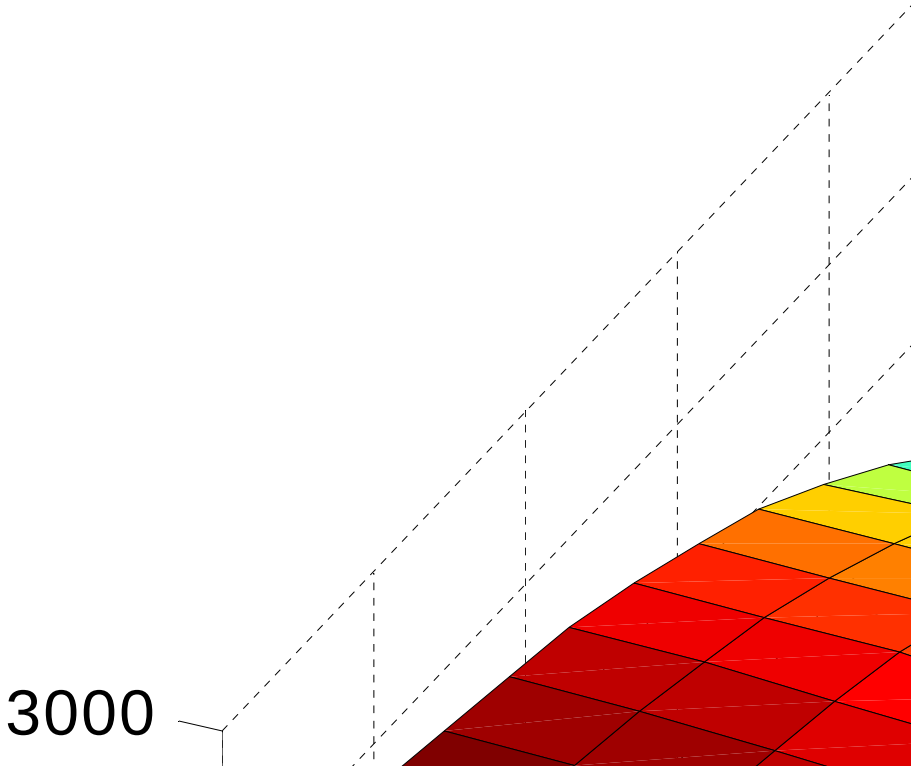
한편, mesh size가 증가함에 따라 spectral radius도 증가하는 것을 확인 할 수 있는데, spectral radius가 점점 1에 근접해지면서 수렴성이 나빠지는 것을 확인 할 수 있다.

그러나 mesh size를 줄이면 좀더 자세한 분포 특성을 확인 할 수 있는 장점이 있다.

>> [A,b]=lsc2d(5);[xk,knes,rho]=gsisol2d(A,b);getgraph(xk);

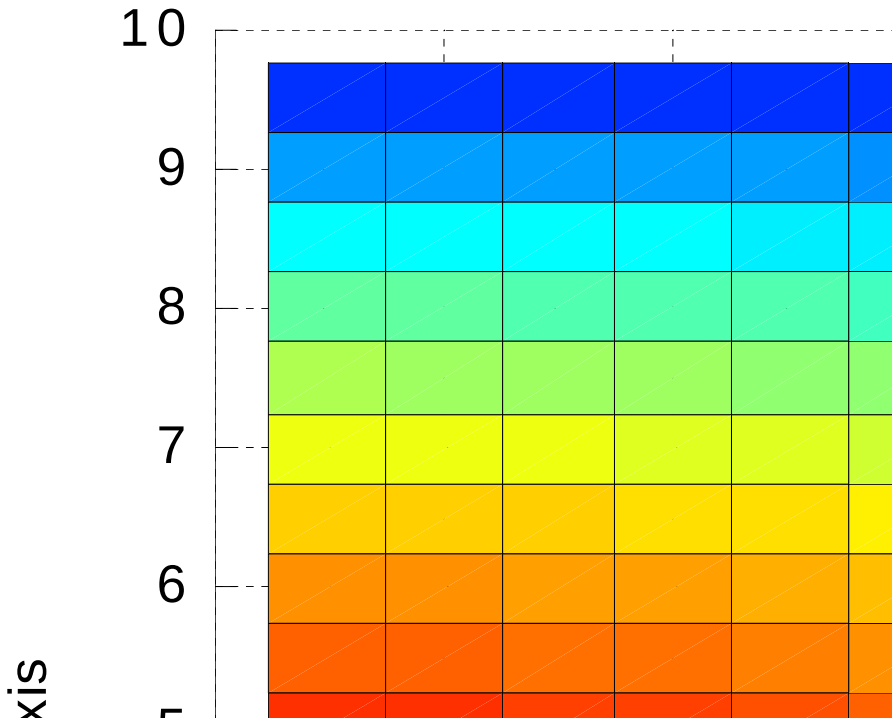


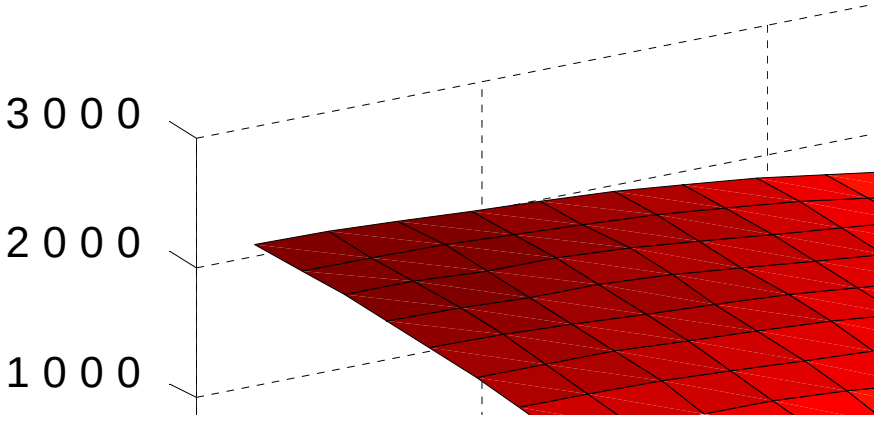

```
>> [A,b]=lsc2d(12);[xk,knes,rho]=gsisol2d(A,b);getgraph(xk);
```



SOR 법을 사용해서 m=20일때의 distribution을 보다 빠르게 얻을 수 있다.

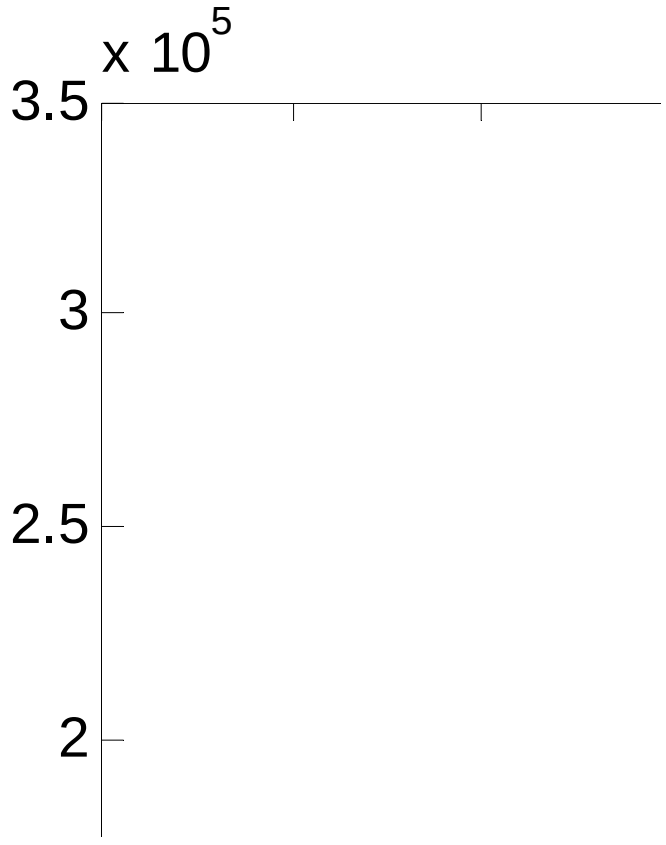
>> m=20;[A,b]=lsc2d(m);[x]=wsor(1.68,A,b);getgraph(x);





직접해법에서 mesh size에 대한 computation 횟수를 비교해보기 위해 다음을 확인하면

```
>> for (m=5:10) [A,b]=lsc2d(m);[x,flops]=lusol2d(A,b);cnt(m-4)=flops;end
>> m=5:10

m =

     5     6     7     8     9    10
>> cnt

cnt =

        5849       16871       41649       91519      183761      343399

>> plot(m,cnt,'k*-')
>>
```



mesh size 10일 때 FLOPS 343399는

```
>> N=10^2;N*(N-1)*(2*N-1)/6+N*(N-1)/2*3

ans =
       343200
```

>>
와 근사적으로 맞는데, 이는 Gauss 소거법의 updating 횟수 + multiplier 계산횟수+ forwad, back substitution 횟수를 더한 값이다. 즉 근사적으로 N^3으로 증가하고, 위의 그래프는 3차 다항식으로 fitting이 될 것으로 예상이 된다.

한편 lusol2d.m 의 마지막 line에 다음을 추가하면 LU operation 시에 필요한 nonzero entry 저장 크기를 구할 수 있다.

nonzr=nnz(U)+nnz(L)+nnz(P) %뒤에 붙어있는 세미콜론을 지우면 바로 출력이 됨

```
>> [A,b]=lsc2d(10);nnz(A)
ans =
    460
>> [xk,cnt]=lusol2d(A,b);
nonzr =

          2118
>>
```

sparse matrix를 사용하여 memory를 아낀다고 하더라도 Gauss 소거시에 발생하는 fill-in 때문에 필요한 기억장소가 m^3으로 증가한다.

```
>>  m=5;[A,b]=lsc2d(m);nnz(A)
[xk,cnt]=lusol2d(A,b);2*m^3
ans =
    105
nonzr =
    283
ans =
    250
>> m=6;[A,b]=lsc2d(m);nnz(A)
[xk,cnt]=lusol2d(A,b);2*m^3
ans =
    156
nonzr =
    478
ans =
    432
>> m=7;[A,b]=lsc2d(m);nnz(A)
[xk,cnt]=lusol2d(A,b);2*m^3
ans =
    217
nonzr =
```

```
    747
ans  =
    686
>>  m=8;[A,b]=lsc2d(m);nnz(A)
[xk,cnt]=lusol2d(A,b);2*m^3
ans  =
    288
nonzr  =
          1102
ans  =
          1024
>>
```

따라서 직접해법의 operation에서 요구되는 기억장소는 natural odering일 경우 m^3의 fill-in에 관련됨을 알 수 있다.

<부록> 함수 리스트

lsc2d.m

lusol2d.m

getgraph.m

gsisol2d.m

eigtgsi.m

sorsol2d.m

wgsiter.m

cmpgsisor.m

vmsgsi.m

wsor.m