AME 535 a Project 3.

1. Jacobi scheme: solve for $\frac{\partial u}{\partial t} = u_{xx} + \cancel{f} + u_{yy} + f$

use finite difference method
$$\frac{u_{i,j}^{r+1} - u_{i,j}^{r}}{\Delta t} = \frac{u_{i+1,j}^{r} - 2u_{i,j}^{r} + u_{i-1,j}^{r}}{h^2} + \frac{u_{i,j+1}^{r} - 2u_{i,j}^{r} + u_{i,j-1}^{r}}{h^2} + f_{i,j}$$

$$\Rightarrow \frac{u_{i,j}^{r+1} - u_{i,j}^{r}}{\Delta t} = \frac{u_{i+1,j}^{r} + u_{i-1,j}^{r} + u_{i,j+1}^{r} + u_{i,j-1}^{r} - 4u_{i,j}^{r}}{h^2} + f_{ij}.$$

Since stability indicates that $\Delta t \le \frac{h^2}{2}$ then take $\Delta t = h^2/2$

$$u_{ij}^{r+1} = \frac{2}{2}u_{i,j} + (u_{i+1,j}^{r} + u_{i-1,j}^{r} + u_{i,j+1}^{r} + u_{i,j-1}^{r} - 4u_{i,j}^{r}) \cdot \frac{1}{\cancel{h^2}} \cdot \frac{\cancel{h^2}}{2} + \frac{h^2}{2} \cdot f_{ij}.$$

$$\Rightarrow u_{i,j}^{r+1} = \frac{1}{2}(u_{i+1,j}^{r} + u_{i-1,j}^{r} + u_{i,j+1}^{r} + u_{i,j-1}^{r} - 2u_{i,j}^{r} + h^2 f_{i,j})$$

which is the algebra form of the scheme.

matrix form: $u^{r+1} = (\underbrace{I - D^{-1}A}_{R_J})u^r + D^{-1}f$

where D is diagnal matrix, I is identity matrix.
(elements)

Gauss Seidel scheme:

from its definition, we will use ~~hade at the r+1~~ the most recently calculated value

thus: $u_{i,j}^{r+1} = \frac{1}{2}(u_{i+1,j}^{r} + u_{i-1,j}^{r+1} + u_{i,j+1}^{r} + u_{i,j-1}^{r+1} + h^2 f_{ij})$

for matrix form $u^{r+1} = \underbrace{(D-L)^{-1}U}_{R_{GS}} \cdot u^r + (D-L)^{-1} f.$

where D is diagnal elements matrix, U is upper triangular matrix

L is lower triangular matrix.

2.

a. extropolate the changes

for Jacobi $u^{n+1} = [wR_J + (1-w)I]u^n + w \cdot D^{-1} f$

for Gauss-Seidel $u^{n+1} = [wR_{GS} + (1-w)I]u^n + w(D-L)^{-1} f$.

b. for Jacobi take 0.2 as a step

and create a for loop to plot convergence for $w = 0.2:0.2:1$

for Gauss-Seidel take 0.2 as a step

create a for loop to plot convergence for $w = 0.2:0.2:2$.

set both times of iteration as 5000 times

See the attached code and plot

from the plot we can see:

for Jacobi method: best w is 1 since it converges fastest

Gauss Seidel method the best w is 1.4 since it converges fastest and also with smallest residue.

c. To calculate flux

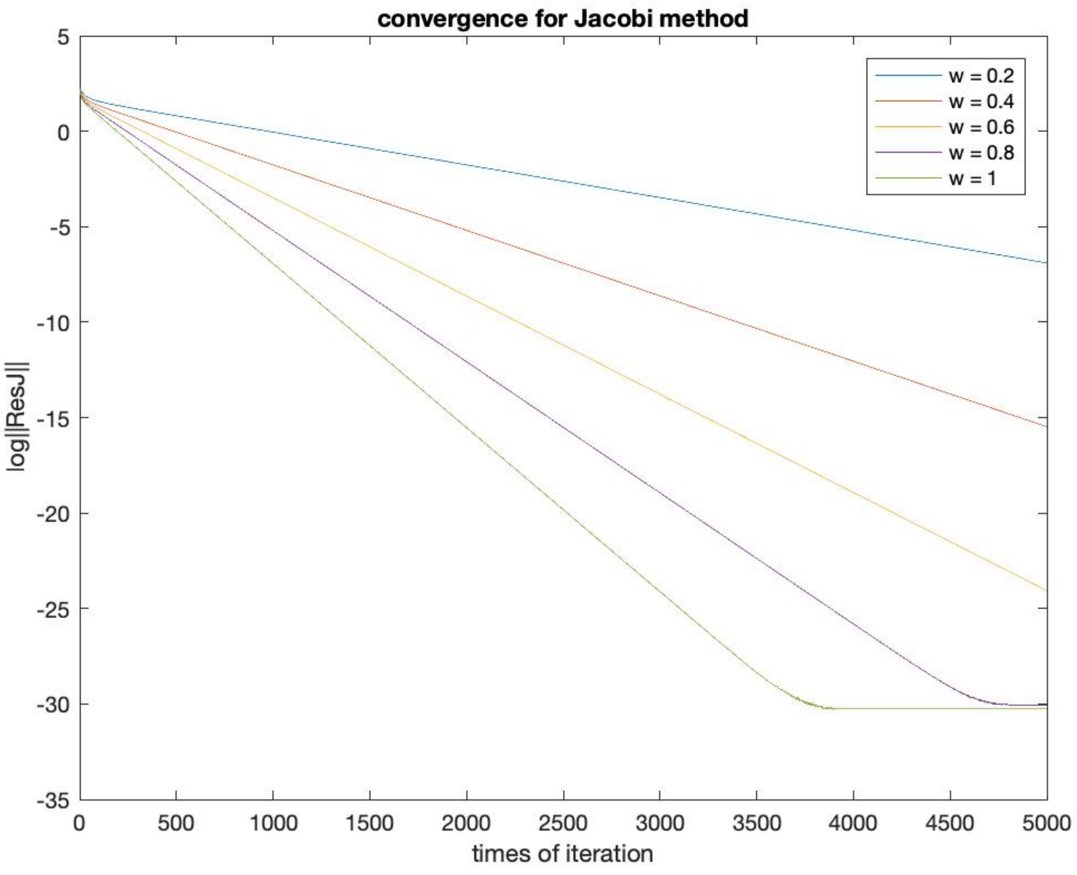
use second order accurate forward & Backward FD method

Bottom, ~~left~~: $\frac{\partial \phi_{1,j}}{\partial n} = \frac{-3\phi_{1,j} + 4\phi_{2,j} - \phi_{3,j}}{2h}$ Left: $\frac{\partial \phi_{i,1}}{\partial n} = \frac{-3\phi_{i,1} + 4\phi_{i,2} - \phi_{i,3}}{2h}$

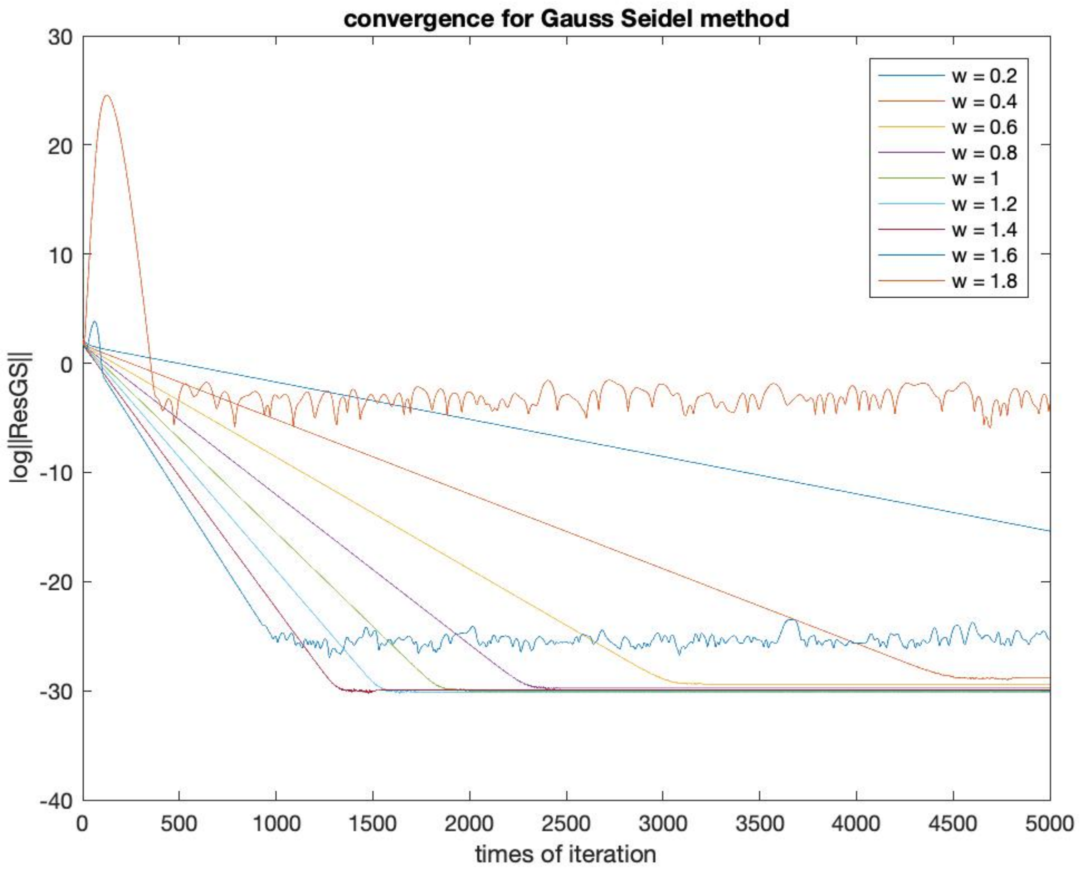
Top: $\frac{\partial \phi_{n,j}}{\partial n} = \frac{-\phi_{n-2,j} + 4\phi_{n-1,j} - 3\phi_{n,j}}{2h}$ Right: $\frac{\partial \phi_{i,n}}{\partial n} = \frac{-\phi_{i,n-2} + 4\phi_{i,n-1} - 3\phi_{i,n}}{2h}$.

See attached code, plot for contour and flux chart. (which is exactly

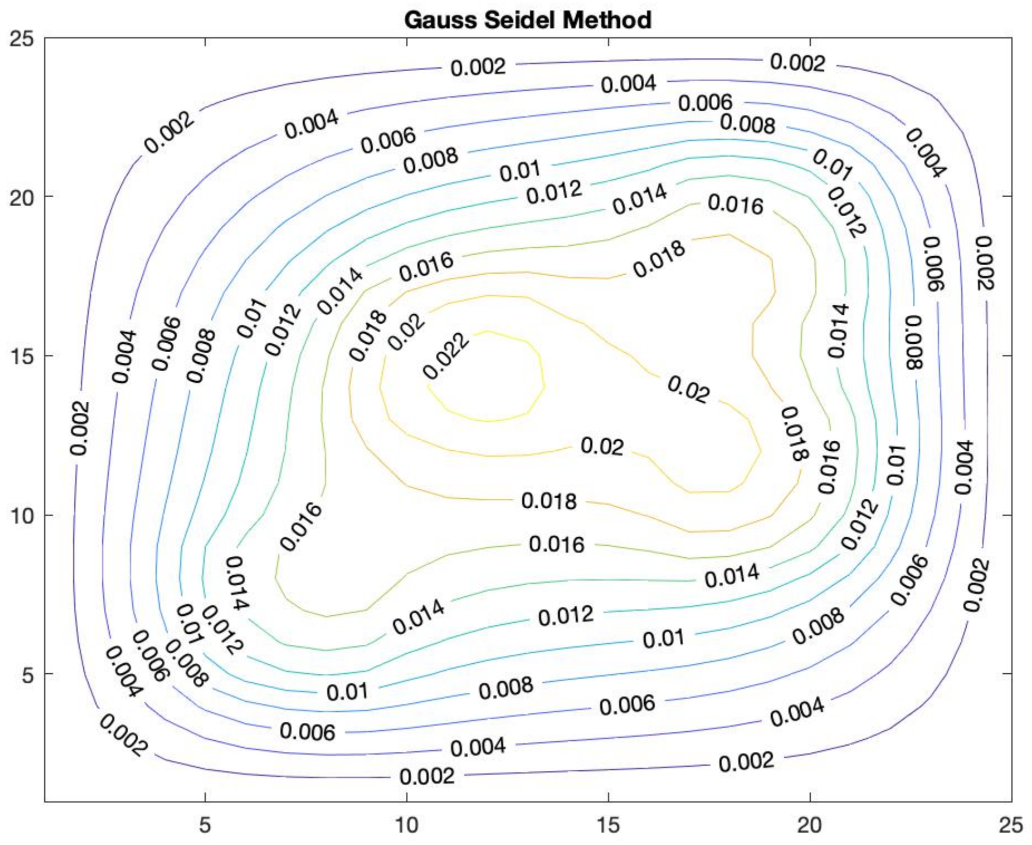
```
%==============question 2.b================
%convergence for Jacobi method
it_num = 1:5000;
for i = 1:5
    res = iteration(25,[1,7,14,16],0.2*i,5000);
    plot(it_num,log(res))
    hold on
end
xlabel('times of iteration')
ylabel('log||ResJ||')
title('convergence for Jacobi method')
legend('w = 0.2','w = 0.4','w = 0.6','w = 0.8','w = 1')
```



```
%convergence for Jacobi method
it_num = 1:5000;
for i = 1:9
    res = iteration(25,[1,7,14,16],0.2*i,5000);
    plot(it_num,log(res))
    hold on
end
xlabel('times of iteration')
ylabel('log||ResGS||')
title('convergence for Gauss Seidel method')
legend('w = 0.2','w = 0.4','w = 0.6','w = 0.8','w = 1','w = 1.2','w = 1.4','w
= 1.6','w = 1.8')
```



```
%calculate flux on boundaries
bot = (-3*solution(1,:)+4*solution(2,:)-solution(3,:))/(2*h);
top = (-solution(n-2,:)+4*solution(n-1,:)-3*solution(n,:))/(2*h);
lft = (-3*solution(:,1)+4*solution(:,2)-solution(:,3))/(2*h);
rit = (-solution(:,n-2)+4*solution(:,n-1)-3*solution(:,n))/(2*h);
%form flux matrix
flux = zeros(n,4);
flux(:,1) = lft;
flux(:,2) = rit;
flux(:,3) = bot';
flux(:,4) = top';
```



Point	Boundary			
	Left	Right	Bottom	Top
1	0.0000	0.0000	0.0000	0.0000
2	0.0122	0.0072	0.0122	0.0054
3	0.0244	0.0145	0.0244	0.0109
4	0.0361	0.0222	0.0360	0.0163
5	0.0466	0.0303	0.0463	0.0217
6	0.0552	0.0391	0.0548	0.0270
7	0.0612	0.0483	0.0605	0.0323
8	0.0643	0.0576	0.0633	0.0373
9	0.0649	0.0662	0.0635	0.0420
10	0.0636	0.0734	0.0618	0.0464
11	0.0611	0.0784	0.0590	0.0504
12	0.0583	0.0811	0.0561	0.0542
13	0.0553	0.0819	0.0533	0.0578
14	0.0522	0.0814	0.0507	0.0612
15	0.0489	0.0803	0.0483	0.0645
16	0.0452	0.0790	0.0459	0.0673
17	0.0411	0.0771	0.0431	0.0689
18	0.0367	0.0740	0.0399	0.0684
19	0.0319	0.0687	0.0360	0.0652
20	0.0268	0.0610	0.0315	0.0589
21	0.0216	0.0509	0.0262	0.0498
22	0.0162	0.0392	0.0202	0.0387
23	0.0108	0.0264	0.0138	0.0262
24	0.0054	0.0132	0.0070	0.0132
25	0.0000	0.0000	0.0000	0.0000

3.

a. Code a program to plot one-grid GS method's convergence.

and a program to plot two-grid GS scheme's convergence.

take relaxation factor as ω_8 for both scheme.

compare to see which is better.

Code a program to plot two-grid GS scheme's convergence

with two different relaxation factor ω_J and ω_8

compare to see which is better. clearly ω_8 is better.

for two grid scheme, iteration time should be calculated as $Cycle \times (V_1 + V_2 + V_c/4)$

take $(V_1, V_2, V_c) = (2, 2, 4)$ for all two grid scheme.

See attached code and plot for convergence.

b. use two grid Gauss Seidel scheme. for $V_c = 2$, $V_c = 4$

set W as ω_8 $V_1 = V_2 = 1$ Time of iteration 500 times

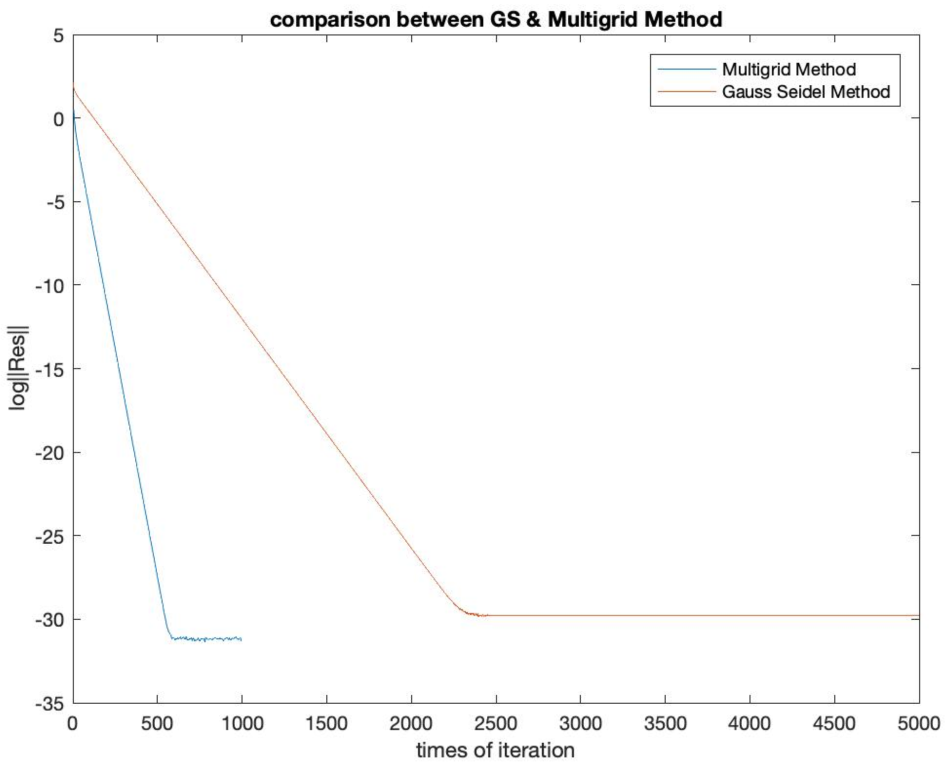
plot convergence, see attached plot

from the plot, we can see $V_c = 4$ converges faster than $V_c = 2$

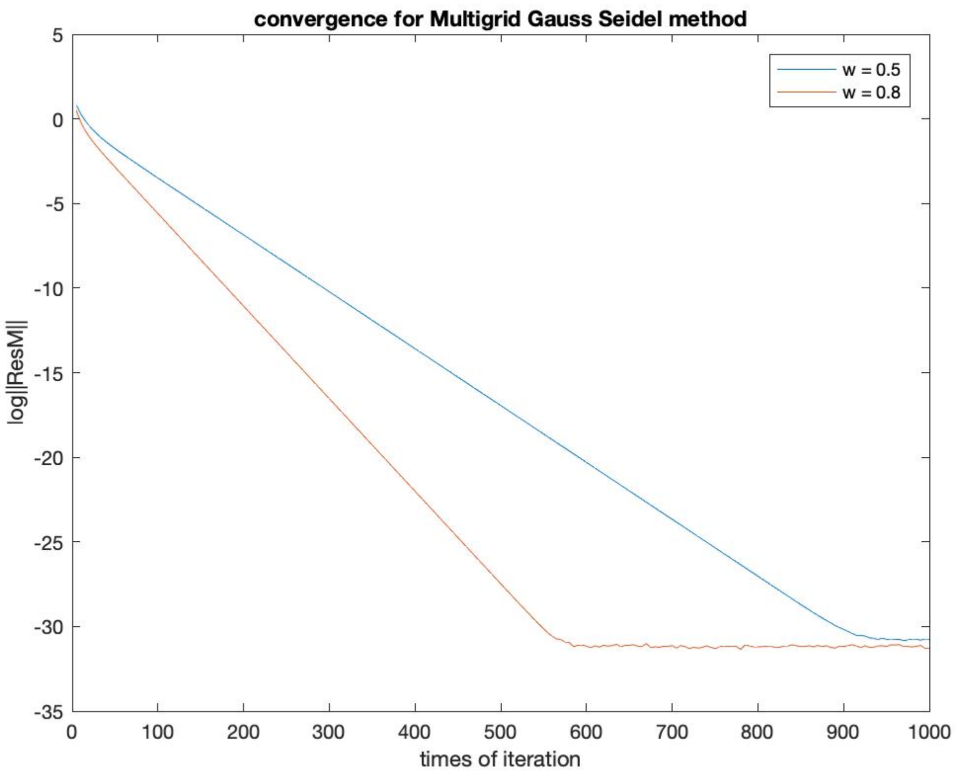
~~but~~ However, to get exact error on coarse grid, one has to do sufficient times of iteration, from est $cycle \times (V_1 + V_2 + V_c/4)$ we can deduce that the convergence for exact error on coarse grid may not be the fastest.

```
%create multigrid solver
function phi = multigridsolver(B,n,w,v1,v2,vc,t)
%create matrix A on fine mesh h
[A,f] = laplace(n,B);

%create A matrix on a coarse grid 2h
nc = (n+1)/2;%size of the coarse grid
[A2,f2] = laplace(nc,B);

%initialize vector phi
phi = zeros(n*n,1);

%implement multigrid method
for time = 1:t
temp = phi;
%pre-smooth for v1 iterations
[phi,res] = GS(phi,n*n,A,f,w,v1);
r = f-A*phi;%compute residue on fine mesh

%restriction
R = reshape(r,n,n);
for i = 1:nc
    temp1(i,:) = R(2*i-1,:);
end
for i = 1:nc
    R2(:,i) = temp1(:,2*i-1);
end
r2 = reshape(R2,nc*nc,1);

%solve for error on coarse grid
e2 = zeros(nc*nc,1);
[e2,res_e] = GS(e2,nc*nc,A2,r2,w,vc);

%prolongation
E2 = reshape(e2,nc,nc);
E = zeros(n,n);
%fill up 2i,2j
for i = 1:nc
    for j = 1:nc
    E(2*i-1,2*j-1) = E2(i,j);
    end
end
%fill up 2i+1,2j
for i = 1:nc
    for j = 1:nc-1
        E(2*i-1,2*j) = (E(2*i-1,2*j-1)+E(2*i-1,2*j+1))/2;
    end
end
%fill up 2i,2j+1
for i = 1:nc-1
    for j = 1:nc
        E(2*i,2*j-1) = (E(2*i-1,2*j-1)+E(2*i+1,2*j-1))/2;
    end
end
%fill up 2i+1,2j+1
for i = 1:nc-1
    for j = 1:nc-1
```

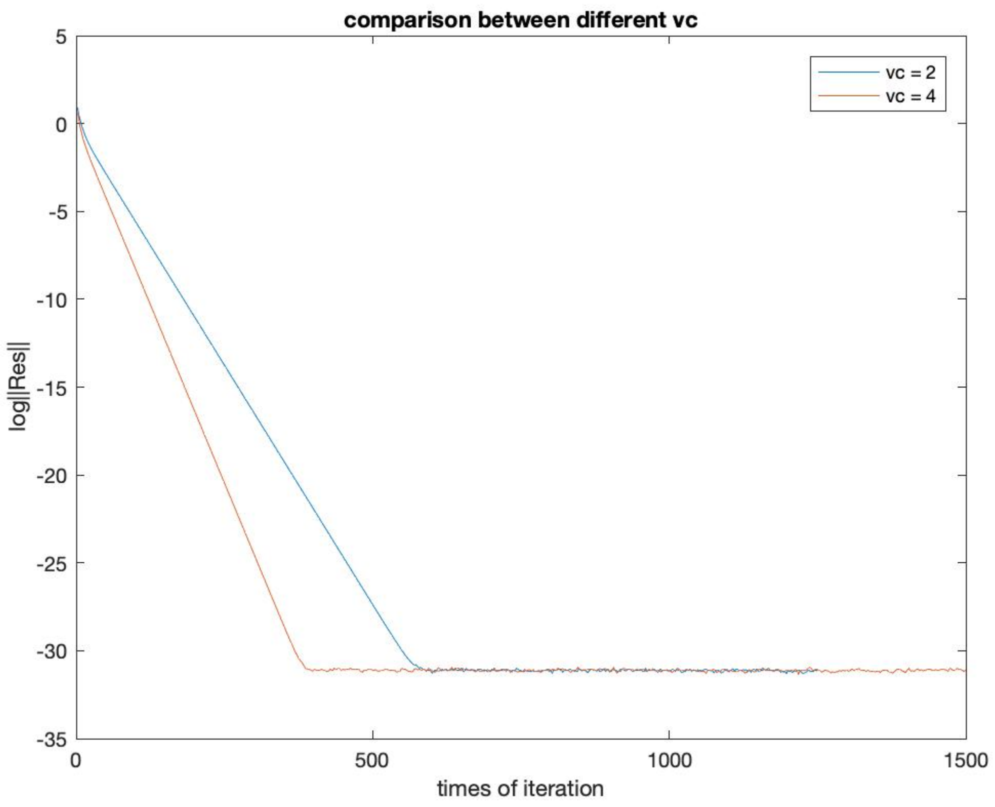
```
        E(2*i,2*j) = (E(2*i-1,2*j-1)+E(2*i+1,2*j-1)+E(2*i-
1,2*j+1)+E(2*i+1,2*j+1))/4;
    end
end

e = reshape(E,n*n,1);

phi = phi+e;

%post-smooth for v2 iterations
[phi,residue(time)] = GS(phi,n*n,A,f,w,v2);%get residue for each cycle
solution = reshape(phi,n,n);
end
```



comparison between GS & Multigrid Method
Multigrid Method
Gauss Seidel Method
log||Res||
times of iteration
5
0
-5
-10
-15
-20
-25
-30
-35
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000



convergence for Multigrid Gauss Seidel method
w = 0.5
w = 0.8
log||ResM||
times of iteration
5
0
-5
-10
-15
-20
-25
-30
-35
0
100
200
300
400
500
600
700
800
900
1000


comparison between different vc
vc = 2
vc = 4
log||Res||
times of iteration
5
0
-5
-10
-15
-20
-25
-30
-35
0
500
1000
1500

4. base on convergence rate

then best method is using two-grid Gauss Seidel Scheme

with $V_1 = V_2 = 2$, $V_c = 4$, $W = 0.8$.

then code to create a loop to try every combination

of 4 elements in [1, 16] and ~~examin~~ examine if the

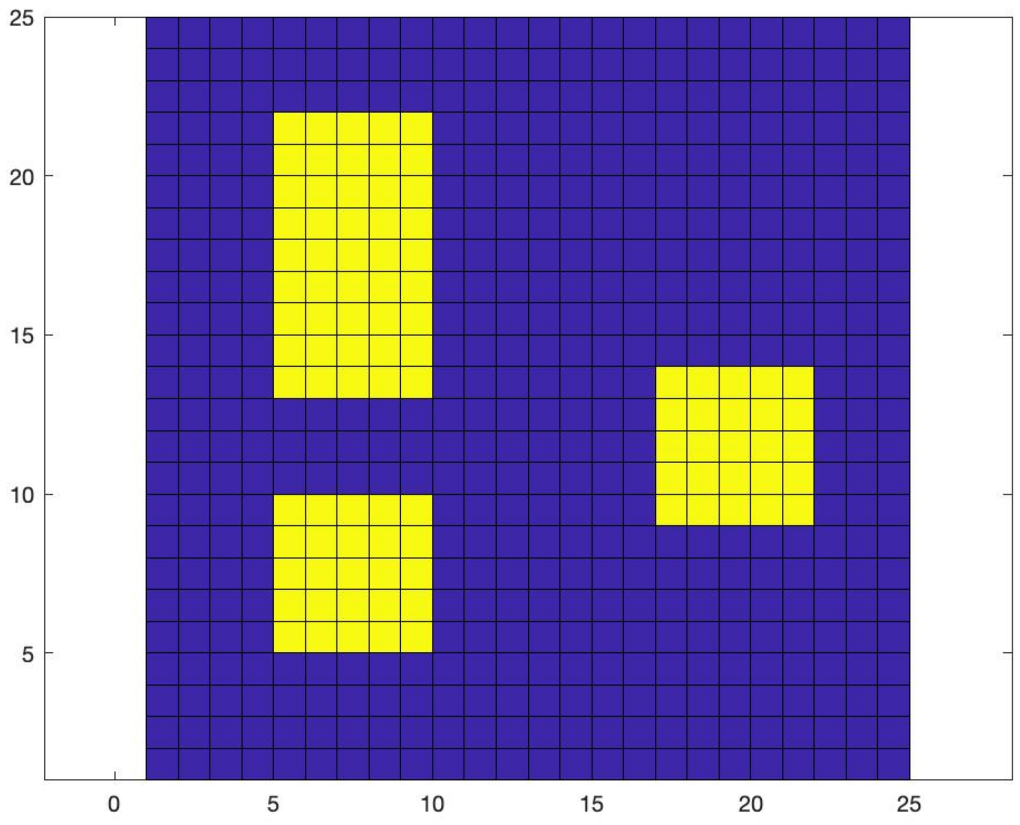
fluxes at boundaries match.

See attached code and ~~flux~~ plot of source matrix.

the result is the number of force block is

[1, 3, 4, 14].

```
%creat loop to calculate force location given unknow boundary flux
flux = load('flux_unknown.mat');
flux=flux.flux;
num = nchoosek(16,4);
time = 0;%number of loop
for i = 1:num
    F = multigridsolver(B(i,:),n,w,v1,v2,t);
    if (abs(F(10,1)-flux(10,1)) < 1e-4 && ...
        abs(F(11,2)-flux(11,2)) < 1e-4 && ...
        abs(F(12,3)-flux(12,3)) < 1e-4 && ...
        abs(F(13,4)-flux(13,4)) < 1e-4)
        source = B(i,:);
        time = time+1;
        break;
    else
        time = time+1;
    end
end
```



5. for V-cycle method

we need to go to the coorsest mesh to calculate errors and then go back to correct potential at each mesh.

Thus we can creat a recursion algorithm. So that the function can ~~implem~~ invoke itself.

To go as high as 4 grid refinements, one has to calculate the finest mesh as $6 \times 2^{4-1} + 1 = 49$ which is 49×49 grid. at least.

So 2 grid $49 \times 49 \rightarrow 25 \times 25$

3 grid $49 \times 49 \rightarrow 25 \times 25 \rightarrow 13 \times 13$

4 grid $49 \times 49 \rightarrow 25 \times 25 \rightarrow 13 \times 13 \rightarrow 7 \times 7$

To compare each scheme set other parameters as

$(B, W, V_1, V_2, V_c) = ([1,7,14,16], 0.8, 1, 1, 1)$

to calculate numbers of iteration, for each scheme, it shall b $(V_1+V_2)_h + \frac{1}{4}(V_1+V_2)_{2h} + (\frac{1}{4})^2 (V_1+V_2)_{4h} + \dots$

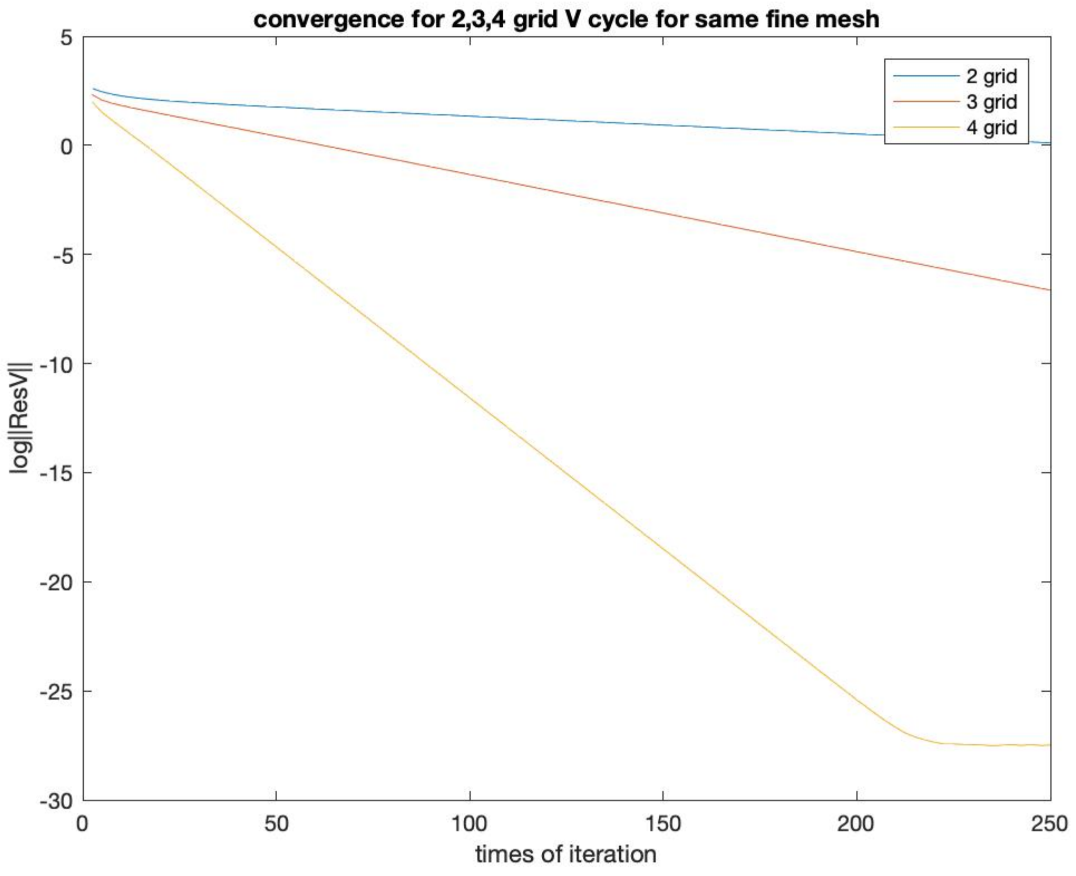
Calculate residue at each iteration

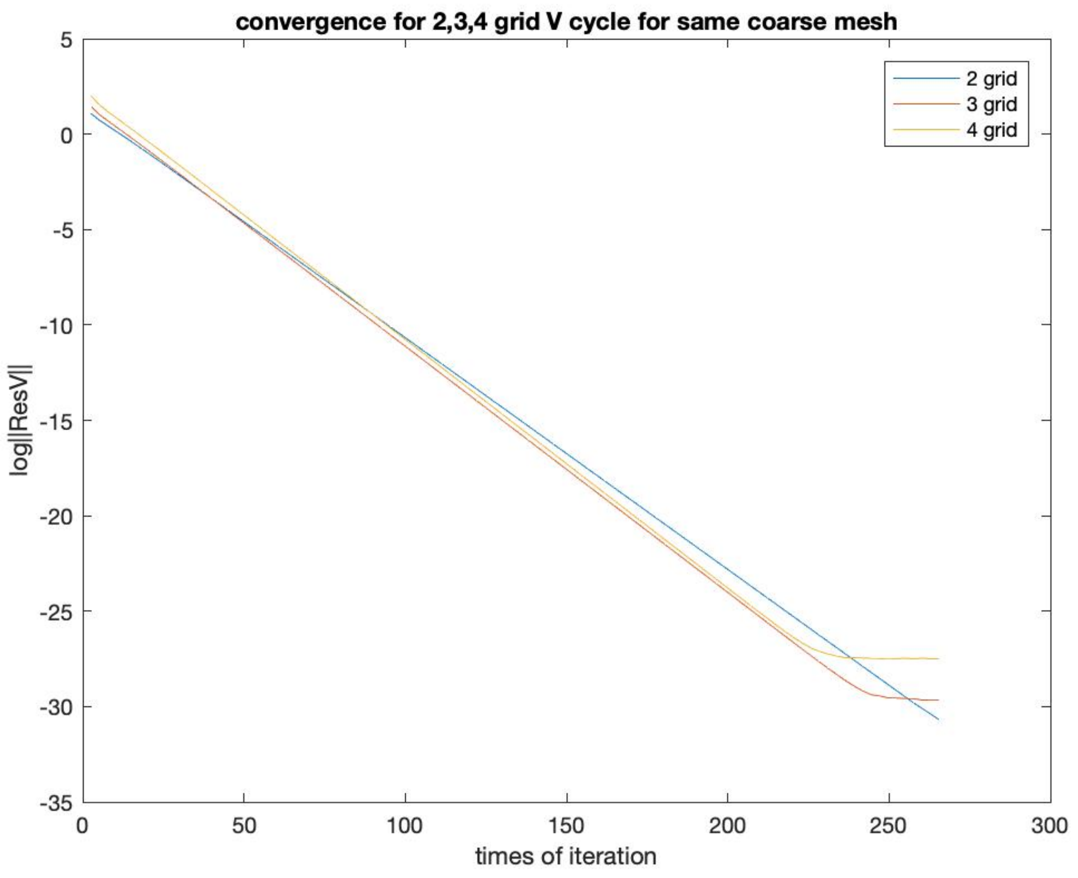
plot the convergence, see attached code and plot.

We can see that for same fine mesh, ~~the~~ 4 grid converges best and it take much more time to convergy for 2 grid

For same coorse mesh, ~~multigrid scheme~~ convergence. ~~for~~ different grid stay almost the same

```
%create vgrid solver
function  phi = vgridsolver(phi,f,n,B,hmin,v1,v2,vc,w)
%create matrix A on fine mesh h
A = laplace(n,B);

if~(n == hmin)
    %pre-smooth for v1 iterations
    phi = GS(phi,n*n,A,f,w,v1);
    r = f-A*phi;%compute residue on fine mesh
    [rc,nc] = restrict(reshape(r,n,n),n);
    ec =
vgridsolver(zeros(nc*nc,1),reshape(rc,nc*nc,1),nc,B,hmin,v1,v2,vc,w);
    [e,n] = prolongate(reshape(ec,nc,nc),nc);
    phi = phi+reshape(e,n*n,1);
    %post-smooth for v2 iterations
    phi = GS(phi,n*n,A,f,w,v2);
else
    phi = GS(phi,n*n,A,f,w,vc);
end

end
```




convergence for 2,3,4 grid V cycle for same coarse mesh
log||ResV||
times of iteration
2 grid
3 grid
4 grid
5
0
-5
-10
-15
-20
-25
-30
-35
0
50
100
150
200
250
300

Appendix: Code

```
%function to create sparse laplacian operater matrix A and forcing
function [A,f] = laplace(n,B)
h = 1/(n-1);%length of the space unit
I = speye(n,n);
E = sparse(2:n,1:n-1,1,n,n);
D = E+E'-2*I;
A = kron(D,I)+kron(I,D);
A = -A./h^2;

Node = zeros(n,n);
N = n*n;%number of nodes
Node(1:N) = 1:N;


%specify boundary condition of matrix A

% ***bottom of the domain***
j = 1;
for i = 2:n-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i, j+1) ) = 0;
   A(ANode_i, Node(i+1, j) ) = 0;
   A(ANode_i, Node(i-1, j) ) = 0;
end

% ***top of the domain***
j = n;
for i = 2:n-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i, j-1) ) = 0;
   A(ANode_i, Node(i+1, j) ) = 0;
   A(ANode_i, Node(i-1, j) ) = 0;
end

% ***left of the domain***
i = 1;
for j = 2:n-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i, j+1) ) = 0;
   A(ANode_i, Node(i, j-1) ) = 0;
   A(ANode_i, Node(i+1, j) ) = 0;
end

% ***right of the domain***
i = n;
for j = 2:n-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i, j+1) ) = 0;
   A(ANode_i, Node(i, j-1) ) = 0;
   A(ANode_i, Node(i-1, j) ) = 0;
end

% domain corners condition
% ***bottom left point***
ANode_i = Node(1,1);
A(ANode_i, Node(2, 1) ) = 0;
A(ANode_i, Node(1, 2) ) = 0;

% ***bottom right point***
ANode_i = Node(1,n);
A(ANode_i, Node(2, n) ) = 0;
A(ANode_i, Node(1, n-1) ) = 0;

% ***top left point***
ANode_i = Node(n,1);
A(ANode_i, Node(n-1, 1) ) = 0;
A(ANode_i, Node(n, 2) ) = 0;

% ***top right point***
ANode_i = Node(n,n);
A(ANode_i, Node(n-1, n) ) = 0;
A(ANode_i, Node(n, n-1) ) = 0;


%create forcing vector
f = zeros(N,1);
ng = (n-1)/6;%size of the grid for each block
for i = 1:4
    b1 = fix(B(i)./4)+1;
    b2 = rem(B(i),4);
    if~(b2 == 0)
        j = 1+b1*ng;%row of the first point of the source block
        k = 1+b2*ng;%column of the first point
        f(Node(k:k+ng,j:j+ng),1) = 1;
    else
        j = 1+(b1-1)*ng;
        k = 1+4*ng;
        f(Node(k:k+ng,j:j+ng),1) = 1;
    end
end

end




%creat Jacobi solver function
function phi1 = Jacobi(phi1,N,A,f,w,t)
Dia = diag(diag(A));
Up = -triu(A,1);
Lo = -tril(A,-1);
RJ = eye(N)-Dia^(-1)*A;%Jacobi iteration matrix
for i = 1:t
    phi1 = (w*RJ+(1-w)*eye(N))*phi1+w*Dia^(-1)*f;
end
end
```

```
%create Gauss-Seidel solver function
function phi2 = GS(phi2,N,A,f,w,t)
Dia = diag(diag(A));
Up = -triu(A,1);
Lo = -tril(A,-1);
RGS = (Dia-Lo)^(-1)*Up;%Gauss-Seidel iteration matrix
for i =1:t
    phi2 = (w*RGS+(1-w)*eye(N))*phi2+w*(Dia-Lo)^(-1)*f;
end
end



%restriction function
function [A,nc] = restrict(M,n)
nc = (n+1)/2;
for i = 1:nc
    temp1(i,:) = M(2*i-1,:);
end
for i = 1:nc
    A(:,i) = temp1(:,2*i-1);
end

end



%prolongation function
function [A,n] = prolongate(M,nc)
n = 2*nc-1;
A = zeros(n,n);
%fill up 2i,2j
for i = 1:nc
    for j = 1:nc
    A(2*i-1,2*j-1) = M(i,j);
    end
end
%fill up 2i+1,2j
for i = 1:nc
    for j = 1:nc-1
        A(2*i-1,2*j) = (A(2*i-1,2*j-1)+A(2*i-1,2*j+1))/2;
    end
end
%fill up 2i,2j+1
for i = 1:nc-1
    for j = 1:nc
        A(2*i,2*j-1) = (A(2*i-1,2*j-1)+A(2*i+1,2*j-1))/2;
    end
end
%fill up 2i+1,2j+1
for i = 1:nc-1
    for j = 1:nc-1
        A(2*i,2*j) = (A(2*i-1,2*j-1)+A(2*i+1,2*j-1)+A(2*i-
1,2*j+1)+A(2*i+1,2*j+1))/4;
    end
end

end
```