University of Science
Faculty of Mathematics and Computer Science

Worksheet #4

Professor: *Laurence Halpern - Juliette Ryan*

Student name: *Nguyen Tu Huy*

Course: *PUF - High Performance Computing*
Due date: *December 08th, 2022*

This workksheet consists in solving with a **Schwarz Domain Decomposition** technique the following 2D classical problem :

$$\begin{aligned} -\Delta u &= f \quad \text{on } \Omega \\ u &= u_D \quad \text{on } \partial\Omega \end{aligned}$$

(f, u_D) chosen so that the exact solution to this problem is the following :

$$u(x) = \sin(\omega x + cx) \times \sin(\omega y + cy) + x^2 + y^2$$

The Laplace operator is discretized by 2nd order centered Finite Difference on a regular cartesian mesh with step hx, hy.

1. Work plan

1. Set up the **Schwarz Domain Decomposition** technique with **Dirichlet interface boundary conditions**.
2. Study numerically (experimentally) the influence of the size of the overlap and compare with theoretical results
3. Do the same with **Neuman boundary conditions**
4. Implement the following Robin interface boundary conditions $\alpha u + \dfrac{\partial u}{\partial n}$
5. Study numerically the influence of the size of the overlap as before and the influence of the α coefficient.
6. Write a small report about your results

2. Provided Matlab functions

4 structures are defined to store the different code data.

- Mg : A global structure containing the definition of $\Omega = [a_1, a_2] \times [b_1, b_2]$, and the total number per direction Nx, Ny
- T : A structure defining the topology of the geometrical domain split

  - T.Ndom = Number of sub-domains
  - T.Ndomx = Number of domains in the x direction
  - T.Ndomy = Number of domains in the y direction

- P : A "Problem" structure that contains for each sub-domain the computed solution array, the interface Dirichlet boundary array u_D, the right hand side f.
- M : A local structure describing the geometry of each sub-domain, and lists of points that list points that will send values to the neighbour domain and points that will receive values from these neighbour domains.

2.1. para2D. First lines of your main program to be completed. It contains

- Geometry
- Discretisation
- Splitting Topology

local matrices, right hand side, ... Your work is to add the rest

$$[P] = \text{schwarz}(P, M, T)$$

that returns P.usol

2.2. mcreamesh. Function that fills in the M structure : Number of points per each subdomain, number and list of neighbours, number and list of points that send and receive data from the neighbour domains.

```
function [M,T] = mcreamesh(Mg, M, T, i)

global drecs2            % Number of overlap cells
drec = 2* drecs2 ;
drec1 = drec + 1 ;

% Splitting Topology
%-------------------------------------------
Ndom = T.Ndom;
Ndomx = T.Ndomx ;
Ndomy = T.Ndomy ;
ix = T.I(i,1)  ;
iy = T.I(i,2) ;


% Number of points per domain and step size hx and hy
%---------------------------------------------------
npmx = floor((Mg.Nx - 1)/Ndomx) ;  % Number of points per domain (as
    balanced as possible)
nrx = Mg.Nx -1 - npmx*Ndomx;
M.Nx = npmx + 1 + (ix <=nrx) ;

npmy = floor((Mg.Ny - 1)/Ndomy) ;   % Number of points per domain (as
    balanced as possible)
nry = Mg.Ny -1 - npmy*Ndomy;
M.Ny = npmy + 1 + (iy <=nry) ;
```

```
25
26 hx = (Mg.a2 - Mg.a1)/(Mg.Nx-1) ;   % Common stepsize
27 % otherwise problem with normal vector
28 hy = (Mg.b2 - Mg.b1)/(Mg.Ny-1) ;   % Common stepsize
29 % otherwise problem with normal vector
30
31 %Split  [a1,a2] x [b1,b2] with no overlap
32 %-----------------------------------------------------
33 M.a1 = Mg.a1 ;
34 i0=0 ;
35 for j=1:ix-1
36     M.a1 = M.a1 + hx*( npmx  + (j <=nrx) ) ;
37     i0=i0 + npmx  + (j <=nrx) ;
38 end
39 M.a2 = M.a1 + hx*(M.Nx-1) ;
40
41 M.b1 = Mg.b1 ;
42 j0 = 0 ;
43 for j=1:iy-1
44     M.b1 = M.b1 + hy*( npmy  + (j <=nry) ) ;
45     j0=j0 + npmy  + (j <=nry) ;
46 end
47 M.b2 = M.b1 + hy*(M.Ny-1) ;
48
49 T.i0 = i0 ;
50 T.j0 = j0 ;
51
52 % Overlap
53 %------------------
54 if  (Ndom > 1)
55     if(ix > 1)
56         M.a1 = M.a1-drecs2*hx ;
57         M.Nx = M.Nx +drecs2 ;
58     end
59     if(ix < Ndomx)
60         M.a2 = M.a2+drecs2*hx;
61         M.Nx = M.Nx +drecs2 ;
62     end
63 end
64
65
66 %Mesh
67 M.hx = hx ;
68 M.x(1:M.Nx) = M.a1 + (0:1:M.Nx-1)*M.hx ;
69
70 M.hy = hy ;
71 M.y(1:M.Ny) = M.b1 + (0:1:M.Ny-1)*M.hy ;
72                                %   l13   l14      l15      l16
73 M.Ndom = Ndom ;                %     -------------------------
74 M.idom = i ;                   %    |      |        |        |
75 mm = M.Nx*M.Ny ;              %    |      |        |        |
76 mx = M.Nx;                     %    |      |        |        |
77 my = M.Ny ;                    %    |      |        |        |
78 icx = 1;                       %    |      |        |        |
79 icy = mx ;                     %    |      |        |        |
80 l1 = 1 ;                       %    |      |        |        |
81 l2 = drec1 ;                   %    |      |        |        |
82 l3 = mx-drec ;                 %    |      |        |        |
```

```
l4 = mx ;                          %    |     |         |         |
                                   %     -------------------------
                                   %   l1    l2        l3        l4
l13 = mm - mx+1 ;
l14 = l13 + drec ;
l15 = mm - drec ;
l16 = mm ;

% % Dirichlet points and interface points
M.Ndir = 0 ;
M.Nvois = 0 ;
M.Nintr = 0 ;
M.lvoisr(1) = 0 ;
M.Ninte = 0 ;
M.lvoise(1) = 0 ;
if(M.Ndom == 1)
    % Dirichlet points
    M.Ndir = 2*mx + 2*(my-2);
    M.ldir(1:mx) = (l1 : icx : l4) ;
    M.ldir(mx + 1: mx + my - 2) = (l4+icy : icy : l16-icy) ;
    M.ldir( mx + my - 1: 2*mx + my-2 ) = ( l13 : icx : l16) ;
    M.ldir( 2*mx + my - 1 : 2*mx + 2*(my-2)) = (l1+icy : icy : l13-icy)
   ;
else
    % % Dirichlet points and interface points
    if(iy == 1) % bottom points
        M.ldir(M.Ndir+1:M.Ndir+mx) = (l1: icx :l4) ;
        M.Ndir = M.Ndir + mx ;
    end
    if(ix == Ndomx) % bottom points
        M.ldir(M.Ndir+1 : M.Ndir + my-2) = (l4 + icy : icy : l16-icy) ;
        M.Ndir = M.Ndir + my-2 ;
    end
    if(iy == Ndomy)
        M.ldir(M.Ndir+1: M.Ndir + mx) = ( l13: icx :l16) ;
        M.Ndir = M.Ndir + mx ;
    end
    if(ix == 1)
        M.ldir(M.Ndir+1 : M.Ndir + my-2) = (l1+icy : icy : l13-icy) ;
        M.Ndir = M.Ndir + my-2 ;
    end
    % Overlap Points send and receive
    %---------   Face South ----------------------------------
    % DIRICHLET

    %---------  Fin  Face South ----------------------------------

    %---------  Face East ----------------------------------
    if(ix < Ndomx)
        M.Nvois = M.Nvois + 1 ;
        nv = (iy-1)*Ndomx + ix +1 ;
        M.novois(M.Nvois) = nv ;
        M.list(nv) = M.Nvois ;

        % reception l4,l16
        % send     l3,l15

        M = FaceInt(M,l4,l16,l3,l15,icy, -icx,1,0) ;
```

```
140
141     end
142     %---------  End  Face Est -----------------------------------
143
144     %---------  Face Nord ----------------------------------
145     % DIRICHLET
146     %---------  End  Face Nord -----------------------------------
147
148     %---------  Face West----------------------------------
149     if(ix > 1)
150         M.Nvois = M.Nvois +1 ;
151         nv = (iy-1)*Ndomx + ix -1 ;
152         M.novois(M.Nvois) = nv ;
153         M.list(  nv  ) = M.Nvois ;
154
155         % reception l1,l13
156         % Send     l2,l14
157
158         M = FaceInt(M,l1,l13,l2,l14,icy,icx,-1,0) ;
159
160     end
161
162     %---------  End  Face Ouest -----------------------------------
163
164 end
165
166 % List of inner points
167 M.Nin = (mx-2)*(my-2);
168 Lin = ones(1,mx*my) ;
169 if(M.Nintr ~= 0)  Lin(M.lintr) = 0 ; end
170 if(M.Ndir ~= 0)  Lin(M.ldir) = 0 ; end
171 num = (1:1:mx*my)  ;
172 M.in = num(Lin==1) ;
173 %
```

2.3. FaceInt. Function called by *mcreamesh* to define normal vectors and interface descriptors.

```
1  function M = FaceInt(M,la,lb,lc,ld,ic1,ic2,nx,ny)
2
3  %Reception face
4  %-------------------------------
5  lab = (lb-la)/ic1 -1  ;
6  M.lintr(M.Nintr+1: M.Nintr +lab) = (la+ic1:ic1:lb-ic1) ;
7
8  % Outward Normal
9  M.nxr(M.Nintr+1: M.Nintr +lab) = nx ;
10 M.nyr(M.Nintr+1: M.Nintr +lab) = ny ;
11
12 M.Nintr = M.Nintr + lab;
13 M.lvoisr(M.Nvois+1) = M.lvoisr(M.Nvois) +lab;
14
15
16 %Send face
17 %-------------------------------
18 lcd = (ld-lc)/ic1 -1 ;
19 M.linte(M.Ninte+1: M.Ninte + lcd) = (lc+ic1:ic1:ld-ic1) ;
20
```

```
% Inward Normal
M.nxe(M.Ninte+1: M.Ninte + lcd) = -nx ;
M.nye(M.Ninte+1: M.Ninte + lcd) = -ny ;

M.Ninte = M.Ninte + lcd;
M.lvoise(M.Nvois+1) = M.lvoise(M.Nvois) + lcd;
```

2.4. init. Function that initialises the numerical solution, the exact solution, the *Dirichlet boundary condition* on $\partial\Omega$, the right hand side (**sm0**).

```
function [sm0, udir, uana, usol] = init(M,Mg,icase)

m1 = M.Nx ;
m2 = M.Ny ;
mm = M.Nx*M.Ny ;
x = M.x ;
y = M.y ;
[xx,yy] = meshgrid(M.x,M.y) ;
xx = xx' ;
yy = yy' ;

switch icase
    case 1  % u = sin(om*x + cx)*sin(om*y + cy)+ x*x + y*y
        omx = 2*pi/(Mg.a2 - Mg.a1) ;
        cx = -omx * Mg.a1 ;
        ux = sin(omx * x' + cx) ;

        omy = 2*pi/(Mg.b2 - Mg.b1) ;
        cy = -omy * Mg.b1 ;
        uy = sin(omy * y' + cy) ;

        uu = ux *uy' ;
        uana = reshape(uu + xx.*xx + yy.* yy,mm,1) ;
        sm0 = reshape( (omx*omx+ omy*omy)*uu - 4 ,mm,1) ;

    case 2  % u = x + y ;
        uana = reshape(xx + yy,mm,1)  ;
        sm0 = zeros(mm,1) ;
    case 22  % u = x ;
        uana = reshape(xx,mm,1)  ;
        sm0 = zeros(mm,1) ;

    case 3  % u = x*x + y*y ;
        uana = reshape(xx.*xx + yy.* yy,mm,1)  ;
        sm0 = -4.*ones(mm,1) ;

    otherwise
        disp(['unknown case' ])
        sm0 = 0 ;
        uana= 0 ;
        udir = 0;
end
udir = zeros(mm,1) ;

if(M.Ndir ~= 0)
    udir(M.ldir) = uana(M.ldir) ;
    sm0(M.ldir) = uana(M.ldir) ;    % apply Dirichlet boundary condition
    to sm0
```

```
48 end
49 usol = zeros(mm,1) ;
```

2.5. creamat. Function that defines the Laplace operator matrix +Dirichlet boundary conditions (ldir + lintr) ($A(i,i) = 1,\ A(i,j) = 0,\ j \neq i$ point *Dirichlet*)

```
1  function [A] = creamat(M)
2
3  % A Laplace Matrix + Dirichlet boundary conditions
4
5  m1 = M.Nx ;
6  m2 = M.Ny ;
7  mm = m1*m2 ;
8
9  e = ones(mm,1);
10 udx2 = 1./(M.hx*M.hx) ;
11 udy2 = 1./(M.hy*M.hy) ;
12 c2 = e*udy2 ;
13 c1 = e*udx2 ;
14 c0 = 2*(c1 + c2);
15 A = spdiags([-c2 -c1 c0 -c1 -c2], ...
16             [-m1 -1 0 1 m1 ], mm, mm) ;
17
18 for i = 1:M.Ndir
19     ii = M.ldir(i) ;
20     A(ii,1:mm) = 0. ;
21     A(ii,ii) = 1. ;
22 end
23
24
25 %% Add loop to take into account  Dirichlet interface conditions
26 %............
27 for i = 1:M.Nintr
28     ii = M.lintr(i) ;
29     A(ii,1:mm) = 0.  ;
30     A(ii,ii) = 1. ;
31 end
```

2.6. visu. Function that visualises mesh or solution

```
1  function visu(indv,T,M,P)
2
3
4  if(indv == 1)
5      figure(1) ;
6      hold on
7      hidden off
8      for i=1:T.Ndom
9          m1=M(i).Nx ;
10         m3=M(i).Ny ;
11         [xx,yy] = meshgrid(M(i).x,M(i).y) ;
12         rng = i/T.Ndom ;
13         on=ones(m1,m3)*rng;
14         mesh(xx',yy',on);view(2)
15     end
16     return
17 end
```

```
18
19 if(indv == 2)
20     figure(2) ;
21     hold on
22     hidden off
23     for i=1:T.Ndom
24         m1=M(i).Nx ;
25         m3=M(i).Ny ;
26         [xx,yy] = meshgrid(M(i).x,M(i).y) ;
27 %         on = reshape(P(i).usol, m1, m3);
28 %         on = reshape(P(i).udir, m1, m3);
29 %         on = reshape(P(i).sm0, m1, m3);
30         on = reshape(P(i).uana, m1, m3);
31         surf(xx',yy',on); view(3)
32     end
33     return
34 end
35
36 if(indv == 3)
37     figure(3) ;
38     hold on
39     hidden off
40     for i=1:T.Ndom
41         m1=M(i).Nx ;
42         m3=M(i).Ny ;
43         [xx,yy] = meshgrid(M(i).x,M(i).y) ;
44         onn = reshape(P(i).usol, m1, m3);
45         surf(xx',yy',onn); view(3)
46     end
47     return
48 end
```

3. Solution

Set up the **Schwarz Domain Decomposition** technique with **Dirichlet interface boundary conditions**.
The problem has form

$$\begin{cases} \Delta u_1^n &= f & \text{in } \Omega_1 \\ u_1^n &= g & \text{in } \partial\Omega \bigcap \partial\Omega_1 \\ u_1^n &= u_2^{n-1} & \text{in } \Gamma_1 \end{cases} \qquad \begin{cases} \Delta u_2^n &= f & \text{in } \Omega_2 \\ u_2^n &= g & \text{in } \partial\Omega \bigcap \partial\Omega_2 \\ u_2^n &= u_1^n & \text{in } \Gamma_2 \end{cases}$$

```
1 function  [P] = schwarz_Jacobi(P,M,T,Mg,epsi,itmax)
2 vari = zeros(T.Ndom,1) ;
3 for it=1:itmax
4     for i=1:T.Ndom
5         P(i).sm = P(i).sm0  ;  % sm0 has the original Dirichlet
     conditions
6                             % on the outer bofders
7         %---------------------------------------
8         %  Updating boundary values
9         %---------------------------------------
10        if(M(i).Nintr~= 0)
11            for j = 1:M(i).Nvois
```

```
                % list of boundary points to be updated
                indSi = (M(i).lvoisr(j)+1 : M(i).lvoisr(j+1));

                indAi = M(i).lintr(indSi) ;

        % List of these points in the neighbour
                idvois = M(i).novois(j) ;
                ij = M(idvois).list(i)  ; % position of these points in
    the neighbour
                indSj = (M(idvois).lvoise(ij)+1 : M(idvois).lvoise(ij+1)
    ) ;
                indAj = M(idvois).linte(indSj) ;
                P(i).sm(indAi) = P(idvois).usol(indAj);
            end
        end
        %---------------------------------------
        %    End  Updating boundary values
        %---------------------------------------
        usol0  = P(i).usol ;                %old solution
        P(i).usol = P(i).A\P(i).sm;         %New solution
        vari(i) = max(abs(usol0-P(i).usol));
    end
    var = max(vari) ;
    sprintf('it= %d  ,var= %e\n', it,var)
%     visu(2,T,M,P) ;
    if( var < epsi ) return;  end

end
```

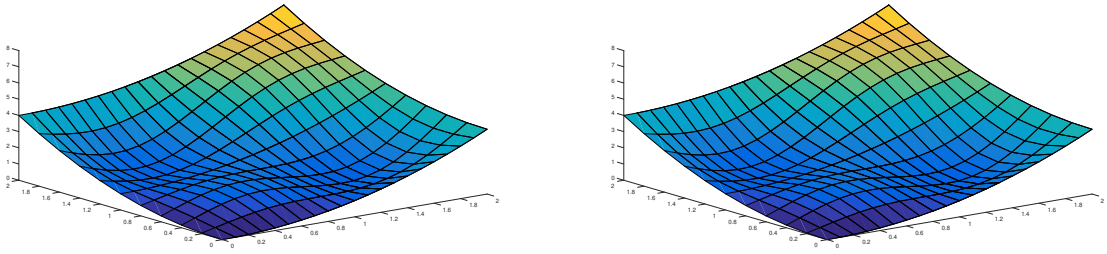
Consider $\Omega = [0, 2] \times [0, 2]$, we split in to 2 subdomain wrt x-axis, and run for $Nx = 101$, $Ny = 80$

Number of overlap cells	Number of iterations	Error norm 2	Error inf-norm
2	102	2.117694e-04	4.280838e-04
3	70	2.119179e-04	4.280838e-04
4	53	2.121793e-04	4.280838e-04
5	43	2.125823e-04	4.280838e-04
6	37	2.125823e-04	4.280838e-04
⋮	⋮	⋮	⋮
49	5	2.993695e-04	4.280838e-04
50	3	2.993695e-04	4.280838e-04
51	3	2.043098e+00	7.870486e+00

Commend:

- The more number of overlap cells is, the less iterations we make to compute. For instance, number of overlap cells is 2, number of iterations is 102. And number of overlap cells is 50, number of iterations just is 3.
- And for the case number of overlap cells = 51, the approximate solution is not close to exact solution.

Consider $\Omega = [0, 2] \times [0, 2]$, we split in to 3 subdomain wrt x-axis, and run for $Nx = 21$, $Ny = 11$



(a) Exact solution (b) Numerical solution

Figure 1: Compare exact solution and numerical solution with Schwarz method, with 3 subdomain and 6 cells overlapping

Number of overlap cells	Number of iterations	Error norm 2	Error inf-norm
2	28	1.251924e-02	1.973941e-02
3	20	1.312092e-02	1.973941e-02
4	15	1.350456e-02	1.973941e-02
5	12	1.380568e-02	1.973941e-02
6	9	1.417079e-02	1.973941e-02
7	15	1.529966e+00	4.446716e+00

- We also have the same result, the more number of overlap cells is, the less iterations we make to compute.
- And for the case number of overlap cells = 7, the approximate solution is not close to exact solution.

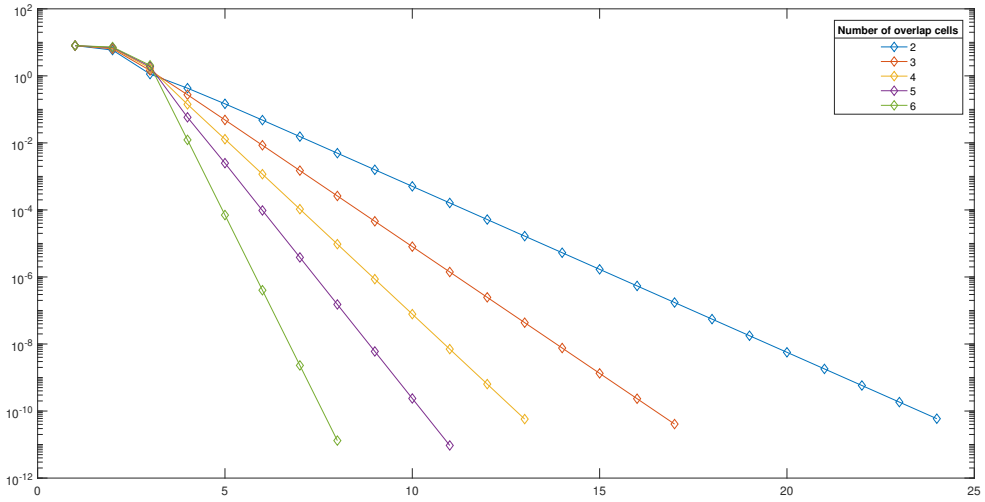


Figure 2: Number of iteration of different number of overlap cells

Conclusion:

- The more number of overlap cells is, the less iterations we make to compute.
- If **number subdomain** $\times$ **number of overlap cells** $\geq$ **number of point** we use to approx at that direction, the approximate solution does not converge to exact solution.

My apply schwarz code for the case subdomain = 2, also work very good and I want you to know what I already done.

```
function [P] = schwarz(P,M,T,Mg,tol,maxit)
it = 1;
error = 1;
err = zeros(1,T.Ndom);
m1 = M.Nx ;
m2 = M.Ny ;
mm = m1*m2;

while it < maxit && error >= tol
    for i = 1:T.Ndom
        P(i).sm = P(i).sm0  ;
        for j = 1:M(i).Ndir          % On Dirichlet boundary condition
            P(i).sm(M(i).ldir(j)) = P(i).udir(M(i).ldir(j));
        end

        for j = 1:M(i).Nintr         % Dirichlet interface conditions
            if i < T.Ndom
                P(i).sm(M(i).lintr(j)) = P(i+1).usol(M(i+1).linte(j));
            else
                P(i).sm(M(i).lintr(j)) = P(1).usol(M(i).linte(j));
            end
        end
        oldsol = P(i).usol ;
        P(i).usol = P(i).A\P(i).sm;
        err(i) = norm(oldsol - P(i).usol);
    end
    it = it + 1;
    error = max(err);
end
end
```

Neuman boundary conditions:

The **Schwarz Domain Decomposition** technique with **Neumann interface boundary conditions**.

The problem has form

$$\begin{cases} \Delta u_1^n &= f & \text{in } \Omega_1 \\ u_1^n &= g & \text{in } \partial\Omega \bigcap \partial\Omega_1 \\ \partial u_1^n &= \partial u_2^{n-1} & \text{in } \Gamma_1 \end{cases} \qquad \begin{cases} \Delta u_2^n &= f & \text{in } \Omega_2 \\ u_2^n &= g & \text{in } \partial\Omega \bigcap \partial\Omega_2 \\ \partial u_2^n &= \partial u_1^n & \text{in } \Gamma_2 \end{cases}$$

On domain 1, we will use the backward Euler difference approximations.

$$\partial u_1^n(i,j) \approx \frac{3u_1^n(i,j) - 4u_1^n(i-1,j) + u_1^n(i-2,j)}{2\Delta x}$$

For the RHS, using central difference approximations , we have

$$\partial u_2^{n-1}(i,j) \approx \frac{u_2^{n-1}(i+1,j) - u_2^{n-1}(i-1,j)}{2\Delta x}$$

On domain 2, we will use the forward Euler approximation of derivatives.

$$\partial u_2^{n}(i,j) \approx \frac{-3u_2^{n}(i,j) + 4u_2^{n}(i+1,j) - u_2^{n}(i+2,j)}{2\Delta x}$$

So we make a change in 2 function:

In matrix A:

```
function [A] = creamatrix(M)

% A Laplace Matrix + Dirichlet boundary conditions

m1 = M.Nx ;
m2 = M.Ny ;
mm = m1*m2 ;

e = ones(mm,1);
udx2 = 1./(M.hx*M.hx) ;
udy2 = 1./(M.hy*M.hy) ;
c2 = e*udy2 ;
c1 = e*udx2 ;
c0 = 2*(c1 + c2);
A = spdiags([-c2 -c1 c0 -c1 -c2], ...
    [-m1 -1 0 1 m1 ], mm, mm) ;

% Boundary condition
for i = 1:M.Ndir
    ii = M.ldir(i) ;
    A(ii,1:mm) = 0. ;
    A(ii,ii) = 1. ;
end


%% Add loop to take into account Neumann interface conditions
% list of interface boundary points to be updated
for j = 1:M.Nvois
    indSi = (M.lvoisr(j)+1 : M.lvoisr(j+1));
    indAi = M.lintr(indSi) ;
    n = length(indAi);

    for k = 1:n
        ii = indAi(k);
        % Vector face take value -1 or 1
        point = M.nxr((j-1)*n + k);
        A(ii,1:mm) = 0. ;
        A(ii,ii)             = point*3./(2*M.hx) ;
        A(ii,ii - point*1)   = point*(-4)./(2*M.hx) ;
        A(ii,ii - point*2)   = point*1./(2*M.hx) ;
%         A(ii,:)
    end

end
```

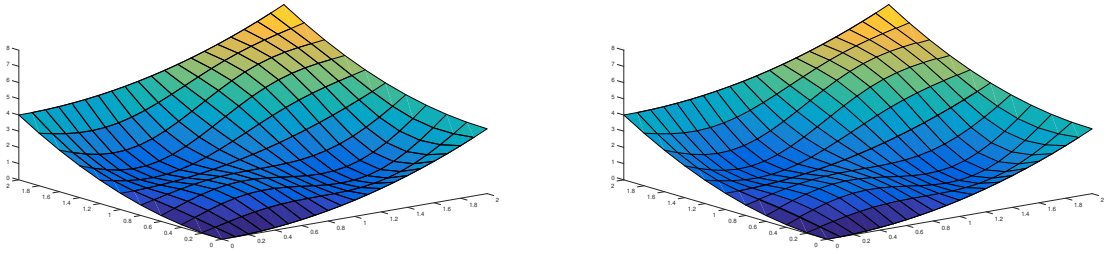
And in function schwarz method:

```
function  [P] = schwarz_Jacobi(P,M,T,Mg,epsi,itmax)
vari = zeros(T.Ndom,1) ;
for it=1:itmax
    for i=1:T.Ndom
        P(i).sm = P(i).sm0  ;  % sm0 has the original Dirichlet
   conditions
                          % on the outer bofders
        %--------------------------------------
        %  Updating boundary values
        %--------------------------------------
        if(M(i).Nintr~= 0)
            for j=1:M(i).Nvois
                % list of boundary points to be updated
                indSi = (M(i).lvoisr(j)+1 : M(i).lvoisr(j+1));
                indAi = M(i).lintr(indSi) ;
        % List of these points in the neighbour
                idvois = M(i).novois(j) ;
                ij = M(idvois).list(i) ;  % position of these points in
   the neighbour
                indSj = (M(idvois).lvoise(ij)+1 : M(idvois).lvoise(ij+1)
   ) ;
                indAj = M(idvois).linte(indSj) ;

                P(i).sm(indAi) = 1./(2*M(idvois).hx)*(P(idvois).usol(
   indAj+1) - P(idvois).usol(indAj-1));
            end
        end
        %--------------------------------------
        %    End Updating boundary values
        %--------------------------------------
        usol0  = P(i).usol ;                %old solution
        P(i).usol = P(i).A\P(i).sm;         %New solution
        vari(i) = max(abs(usol0-P(i).usol));
    end
    var = max(vari) ;
    sprintf('it= %d  ,var= %e\n', it,var)
%     visu(2,T,M,P) ;
    if( var < epsi ) return;  end

end
```

Consider $\Omega = [0, 2] \times [0, 2]$, we split in to 2 subdomain wrt x-axis, and run for $Nx = 21$, $Ny = 11$



(a) Exact solution (b) Numerical solution

Figure 3: Compare exact solution and numerical solution with Schwarz method, with 2 subdomain and 9 cells overlapping

Number of overlap cells	Number of iterations	Error norm 2	Error inf-norm
2	15	2.291869e-02	5.195617e-02
3	11	1.533967e-02	2.603836e-02
4	9	1.109373e-02	2.603836e-02
5	8	1.111174e-02	2.603836e-02
6	7	1.450691e-02	2.674770e-02
⋮	⋮	⋮	⋮
9	6	2.396722e-02	5.052541e-02
10	316	NaN	NaN

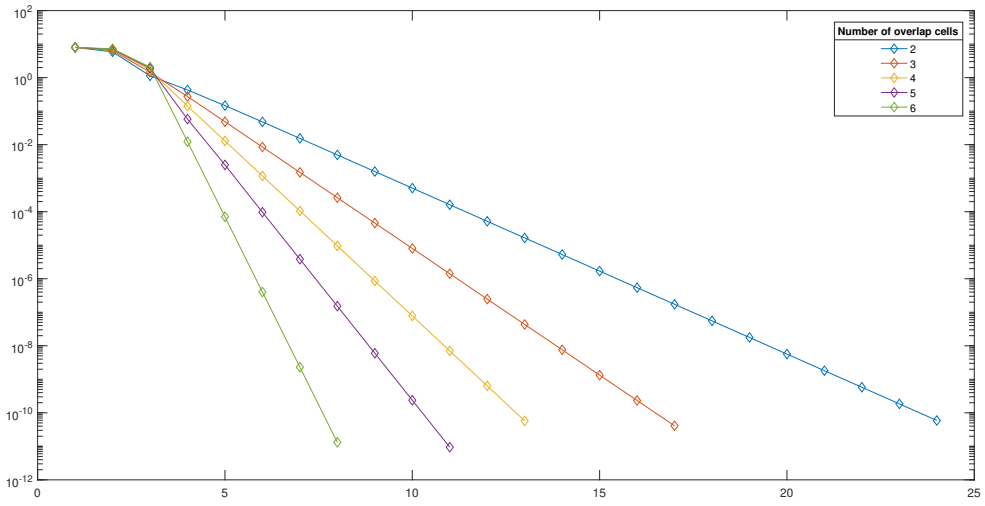


Figure 4: Number of iteration of different number of overlap cells

Commend:

- The more number of overlap cells is, the less iterations we make to compute.

- If **number subdomain** $\times$ **number of overlap cells** $\geq$ **number of point** we use to approx at that direction, the approximate solution does not converge to exact solution.

Robin boundary conditions:

The **Schwarz Domain Decomposition** technique with **Robin interface boundary conditions**.

The problem has form

$$\begin{cases} \Delta u_1^n & = f & \text{in } \Omega_1 \\ u_1^n & = g & \text{in } \partial\Omega \bigcap \partial\Omega_1 \\ \partial u_1^n + \rho u_1^n & = \partial u_2^{n-1} + \rho u_2^{n-1} & \text{in } \Gamma_1 \end{cases} \qquad \begin{cases} \Delta u_2^n & = f & \text{in } \Omega_2 \\ u_2^n & = g & \text{in } \partial\Omega \bigcap \partial\Omega_2 \\ \partial u_2^n + \rho u_2^n & = \partial u_1^n + \rho u_1^n & \text{in } \Gamma_2 \end{cases}$$

On domain 1, we will use the backward Euler difference approximations.

$$\partial u_1^n(i,j) \approx \frac{3u_1^n(i,j) - 4u_1^n(i-1,j) + u_1^n(i-2,j)}{2\Delta x}$$

For the RHS, using central difference approximations , we have

$$\partial u_2^{n-1}(i,j) \approx \frac{u_2^{n-1}(i+1,j) - u_2^{n-1}(i-1,j)}{2\Delta x}$$

On domain 2, we will use the forward Euler approximation of derivatives.

$$\partial u_2^n(i,j) \approx \frac{-3u_2^n(i,j) + 4u_2^n(i+1,j) - u_2^n(i+2,j)}{2\Delta x}$$

I make a change in 2 function:
In matrix A:

```
function [A] = creamat(M,dem,rho)

% A Laplace Matrix + Dirichlet boundary conditions

m1 = M.Nx ;
m2 = M.Ny ;
mm = m1*m2 ;

e = ones(mm,1);
udx2 = 1./(M.hx*M.hx) ;
udy2 = 1./(M.hy*M.hy) ;
c2 = e*udy2 ;
c1 = e*udx2 ;
c0 = 2*(c1 + c2);
A = spdiags([-c2 -c1 c0 -c1 -c2], ...
    [-m1 -1 0 1 m1 ], mm, mm) ;

for i = 1:M.Ndir
    ii = M.ldir(i) ;
    A(ii,1:mm) = 0. ;
    A(ii,ii) = 1. ;
end
```

```
23
24
25 %% Add loop to take into account Robin interface conditions
26
27 if dem == 1 % Backward
28     for i = 1:M.Nintr
29         ii = M.lintr(i);
30         A(ii,1:mm) = 0. ;
31         A(ii,ii) = 3./(2*M.hx) + rho ;
32         A(ii,ii-1) = -4./(2*M.hx) ;
33         A(ii,ii-2) = 1./(2*M.hx) ;
34     end
35 end
36 if dem == 2 % Forward
37     for i = 1:M.Nintr
38         ii = M.lintr(i);
39         A(ii,1:mm) = 0. ;
40         A(ii,ii) = -3./(2*M.hx) + rho;
41         A(ii,ii+1) = 4./(2*M.hx) ;
42         A(ii,ii+2) = -1./(2*M.hx) ;
43     end
44 end
```

And in function schwarz method:

```
1  function  [P] = schwarz_Jacobi(P,M,T,rho,Mg,epsi,itmax)
2  vari = zeros(T.Ndom,1) ;
3  for it=1:itmax
4      for i=1:T.Ndom
5          P(i).sm = P(i).sm0  ;  % sm0 has the original Dirichlet
       conditions
6                              % on the outer bofders
7          %---------------------------------------
8          %  Updating boundary values
9          %---------------------------------------
10         if(M(i).Nintr~= 0)
11             for j=1:M(i).Nvois
12                 % list of boundary points to be updated
13                 indSi = (M(i).lvoisr(j)+1 : M(i).lvoisr(j+1));
14                 indAi = M(i).lintr(indSi) ;
15         % List of these points in the neighbour
16                 idvois = M(i).novois(j) ;
17                 ij = M(idvois).list(i) ;  % position of these points in
       the neighbour
18                 indSj = (M(idvois).lvoise(ij)+1 : M(idvois).lvoise(ij+1)
       ) ;
19                 indAj = M(idvois).linte(indSj) ;
20
21                 P(i).sm(indAi) = 1./(2*M(idvois).hx)*(P(idvois).usol(
       indAj+1) - P(idvois).usol(indAj-1)) + rho*P(idvois).usol(indAj);
22             end
23         end
24         %---------------------------------------
25         %    End Updating boundary values
26         %---------------------------------------
27         usol0  = P(i).usol ;                  %old solution
28         P(i).usol = P(i).A\P(i).sm;           %New solution
29         vari(i) = max(abs(usol0-P(i).usol));
30     end
```

```
    var = max(vari) ;
%     sprintf('it= %d  ,var= %e\n', it,var)
%     visu(2,T,M,P) ;
    if( var < epsi ) return;  end

end
```

Consider $\Omega = [0,2] \times [0,2]$, we split in to 2 subdomain wrt x-axis, and run for $Nx = 21$, $Ny = 11$, $\alpha = 1$

Number of overlap cells	Number of iterations	Error norm 2	Error inf-norm
2	18	2.637060e-02	8.350196e-02
3	13	1.736663e-02	4.689931e-02
⋮	⋮	⋮	⋮
9	6	2.608484e-02	7.265974e-02
10	256	NaN	NaN

Consider $\Omega = [0,2] \times [0,2]$, we split in to 2 subdomain wrt x-axis, and run for $Nx = 21$, $Ny = 11$, $\alpha = 10$

Number of overlap cells	Number of iterations	Error norm 2	Error inf-norm
2	13	1.218029e-02	3.831842e-02
3	10	1.187141e-02	3.379307e-02
⋮	⋮	⋮	⋮
9	4	1.387729e-02	2.281914e-02
10	35	8.925288e-01	4.785770e+00

Consider $\Omega = [0,2] \times [0,2]$, we split in to 2 subdomain wrt x-axis, and run for $Nx = 21$, $Ny = 11$, $\alpha = 100$

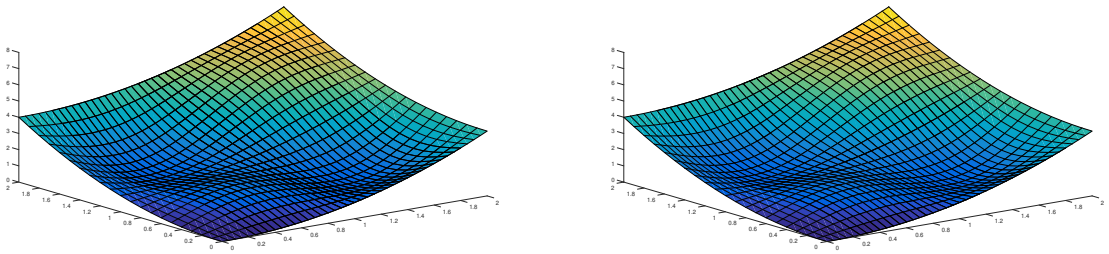
Number of overlap cells	Number of iterations	Error norm 2	Error inf-norm
2	13	1.009070e-02	1.993984e-02
3	10	1.067184e-02	1.983769e-02
⋮	⋮	⋮	⋮
9	4	1.365666e-02	2.011543e-02
10	5	9.925255e-02	2.320671e-01

Commend:

- The more number of overlap cells is, the less iterations we make to compute.
- If **number subdomain** $\times$ **number of overlap cells** $\geq$ **number of point** we use to approx at that direction, the approximate solution does not converge to exact solution.

- If the value α is large, the problem is most likely to **Dirichlet BC**.
- The result of $\alpha = 1$ is not except with number of overlap cells = 10, but for $\alpha = 10$ or $\alpha = 100$ we can accept the value of overlap cells = 10.

For the case, $\Omega = [0, 2] \times [0, 2]$, we split in to 3 subdomain wrt x-axis, and run for $Nx = 41$, $Ny = 21$, $\alpha = 1$



(a) Exact solution (b) Numerical solution

Figure 5: Compare exact solution and numerical solution with Schwarz method, with **3 subdomain** and **9 cells overlapping**

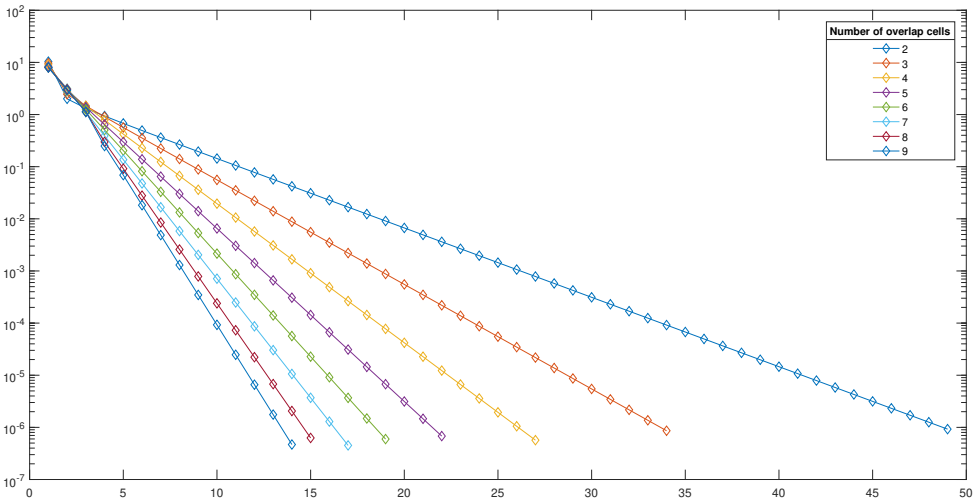


Figure 6: Number of iteration of different number of overlap cells