

1. Introduction. This is a description of how I did alternating, one, and two level Schwarz methods in 2D using a five point difference stencil. My codes are for Poisson's equation

$$-u_{xx} - u_{yy} = f(x, y), \text{ for } 0 < x, y < 1$$

with zero boundary conditions.

1.1. Grids. Both the entire domain and the subdomains will be squares. If the entire domain is an $n \times n$ mesh, then $h = 1/(n + 1)$. The subdomains will have $l \times l$ grids, but h will **remain the same!!**.

I evaluate the matrix-vector product on the full domain with the MATLAB function `lapmf.m` (matrix-free Laplacian) which is formulated for a one dimensional vector, converts it to 2D internally, and then converts the product back to 1D.

```
%
% matrix-free negative Laplacian
%
function lu=lapmf(u)
n2=length(u);
n=sqrt(n2);
h2=(n+1)*(n+1);
%
% transform the linear array into a 2d array
%
uu=zeros(n,n);
uu(:)=u;
%
% compute the discrete negative Laplacian on the 2d array
%
luu=4*uu;
if n>1
luu(1:n-1,:)=luu(1:n-1,:)-uu(2:n,:);
luu(2:n,:)=luu(2:n,:)-uu(1:n-1,:);
luu(:,2:n)=luu(:,2:n)-uu(:,1:n-1);
luu(:,1:n-1)=luu(:,1:n-1)-uu(:,2:n);
end
%
% convert to a linear array, divide by h^2
%
lu=luu(:)*h2;
```

1.2. Subdomains and Overlaps. I'll use 2^p subdivisions in each direction so there will be 2^{p+1} subdomains. Each subdomain will have the same number of interior mesh points.

Here's how to do this for a given overlap. If I want an overlap of o points, I let

$$n = 2^m + o$$

and build an $n \times n$ grid. I split the grid into equal sections of $l = 2^{m-p} + o - 1$ points on a side. The matlab code `partition.m` does this, returning the left and right endpoints of the intervals.

```

function [l,r]=partition(n,p,o)
%
% split the set [1,n] into p intervals with overlap o
% return the left and right endpoints
%
% I assume that n=2^q+o, which allows for a perfect split with
% all intervals having equal length
%
l=zeros(p,1); r=zeros(p,1); base=n-o; len=(base/p)+o-1;
l(1)=1; r(1)=len+1;
for i=1:p-1
    r(i)=l(i)+len;
    l(i+1)=r(i)-o+1;
end
r(p)=l(p)+len;
if (r(p)~=n)
    disp('partition error');
    return
end

```

1.3. Subdomain Solvers. I build the sparse matrix for the subdomains using the MATLAB sparse matrix functions. The function `pmat2.m` builds the Laplacian for an $m_x \times m_y$ mesh with a given value of h .

```

function al=pmat2(mx,my,h)
%
% matrix for 2D Laplacian
%
n=mx*my;
hx=h; hy=h;
h2=h^-2;
offdx=-1*h2*ones(n,1);
offdy=-1*h2*ones(n,1);
dd=4*h2*ones(n,1);
al=spdiags([offdy, offdx, dd, offdx, offdy], [-mx, -1,0,1,mx],n,n);
for i=1:my-1
    p=i*mx;
    al(p,p+1)=0;
    al(p+1,p)=0;
end

```

Since all subdomains are the same size, I can build the subdomain matrix once and use it for all the subdomain solves (this trick does not work in the real world). The MATLAB code `aform` does this.

```

function af=aform(m,ph,pv,overlap)
h=1/(m+1);
[lv,rv]=partition(m,pv,overlap);
if pv~=ph

```

```
        disp('error in afform.m');  
        return  
end  
mx=rv(1)-lv(1)+1;  
af=pmat2(mx,mx,h);
```

Once the subdomain matrix has been build, I factor it once and keep the factors for the entire solve. So in my test programs, `pctst1.m` and `pctst2.m`, I have a line like

```
af=afform(m,ph,pv,overlap); uf=chol(af); lf=uf';
```

which uses the cholesky factorization in MATLAB.

If your problem is non-symmetric need to use an LU factorization, which takes twice as much time.

```
af=afform(m,ph,pv,overlap); [lf,uf]=lu(af);
```

Cholesky is faster and takes less room (if you don't store lf like I do for clarity and generality).

I keep the factors as global variables in MATLAB.

2. One Level Methods.

2.1. One Level Multiplicative. Once I've have the subdomain solvers, a one-level code is easy to build. Here's `pcmulsw.m`, a non-symmetric multiplicative Schwarz preconditioner. Pay attention to how much trouble it is to back and forth from a 2D array, where the subdomains are easy to build, to a 1D array, to which the matrices are applied.

```
%
% multiplicative Schwarz as a preconditioner on an m x m mesh
%
function up=pcmulsw(au)
global ph pv overlap
[m1,m2]=size(au); m=sqrt(m1);
au2=zeros(m,m);
au2(:)=au;
%ph=2; pv=2; overlap=max(3,floor(m/5));
up2=mutswp(m,au2,overlap,ph,pv);
up=up2(:);
%
% This is a modification of the solver. Divide the domain into ph * pv
% subdomains, ph splits in horizontal and pv splits in vertical
%
function u2d=mutswp(m,rhs2d,overlap,ph,pv)
[l,r]=partition(m,2,overlap);
[lv,rv]=partition(m,pv,overlap);
[lh,rh]=partition(m,ph,overlap);
h=1/(m+1);
u2d=zeros(m,m);
maxit=1;
%
% lower block
%
mr=rv(1);
lr=rv(1)-lv(1)+1;
for i=1:maxit
for ih=1:ph
for iv=1:pv
%
% residual update
%
r2d=res2d(u2d,rhs2d);
lbh=lh(ih); ubh=rh(ih); lbv=lv(iv); ubv=rv(iv);
vd=ubv-lbv+1; hd=ubh-lbh+1;
rsub=r2d(lbh:ubh,lbv:ubv);
ux=subdomain(rsub,hd,vd,h);
u2d(lbh:ubh,lbv:ubv)=u2d(lbh:ubh,lbv:ubv)+ux;
end
```

```

end
end
function r2=res2d(uin,rhs2d)
[m1,m2]=size(rhs2d);
if m1 ~= m2
disp('error in res2d');
return;
end
r2=zeros(m1,m1);
uin1d=uin(:); rhs1d=rhs2d(:);
res1d=rhs1d-lapmf(uin1d);
r2(:)=res1d;

```

The preconditioner is not symmetric, so I'll use it with GMRES and send GMRES the (right!) preconditioned Laplacian.

```

%
% preconditioner laplace operator
%
function au=mullap(u)
u1=pcmulsw(u); au=lapmf(u1);

```

I can symmetrize the preconditioner for use with CG by going through the loop backwards. `pcsmulsw.m` does that. Here are the loops that call the subdomain solvers. All the rest is the same as `pcmulsw.m`.

```

for i=1:maxit
    %
    % forward sweep
    %
    for ih=1:ph
        for iv=1:pv
            %
            % residual update
            %
            r2d=res2d(u2d,rhs2d);
            lbh=lh(ih); ubh=rh(ih); lbv=lv(iv); ubv=rv(iv);
            vd=ubv-lbv+1; hd=ubh-lbh+1;
            rsub=r2d(lbh:ubh,lbv:ubv);
            ux=subdomain(rsub,hd,vd,h);
            u2d(lbh:ubh,lbv:ubv)=u2d(lbh:ubh,lbv:ubv)+ux;
        end
    end
    %
    % backward sweep
    %
    for ih=ph:-1:1
        for iv=pv:-1:1
            %
            % residual update

```

```

%
r2d=res2d(u2d,rhs2d);
lbh=lh(ih); ubh=rh(ih); lbv=lv(iv); ubv=rv(iv);
vd=ubv-lbv+1; hd=ubh-lbh+1;
rsub=r2d(lbh:ubh,lbv:ubv);
ux=subdomain(rsub,hd,vd,h);
u2d(lbh:ubh,lbv:ubv)=u2d(lbh:ubh,lbv:ubv)+ux;
end
end
end

```

The testing program is `schwarz_test.m`. The loops sweep through 4×4 , 8×8 , 16×16 , and 32×32 splits using mesh sizes of (roughly) $1/32$, $1/64$, and $1/128$. We terminate when the residual has been reduced by a factor of $1/100$ and use right preconditioning so that the termination is independent of the preconditioner. We tabulate linear iteration counts upon termination.

And now for the results. In each of the 4×3 tables that follow, H is roughly decreased by a factor of two going down the columns and h is similarly decreased going across the rows. The exact values of H and h are functions of the overlap and I'll only put rough values in the tables.

As you can see the symmetric preconditioner with CG requires fewer iterations but the cost of each iteration is more. From the condition estimate $\kappa = 1 + O(1/(Hh))$ and the expectation that the iteration count is proportional to $\sqrt{\kappa}$ you'd expect to the the iteration count double as you do down the diagonal on which H and h are halved simultaneously. It's pretty close!

TABLE 2.1
Iteration Statistics for Symmetric Multiplicative Schwarz with CG

	Overlap = 0			Overlap = 1		
$H \backslash h$	1/32	1/64	1/128	1/32	1/64	1/128
1/4	8	11	16	6	8	12
1/8	9	14	20	7	10	14
1/16	12	17	27	7	11	18
1/32	15	22	35	8	14	22
	Overlap = 1%			Overlap = 2%		
1/4	6	8	10	6	7	8
1/8	7	10	12	7	8	10
1/16	7	11	14	7	9	12
1/32	8	14	17	8	10	14

TABLE 2.2
Iteration Statistics for Multiplicative Schwarz with GMRES

	Overlap = 0			Overlap = 1		
$H \setminus h$	1/32	1/64	1/128	1/32	1/64	1/128
1/4	9	12	16	7	9	12
1/8	12	16	22	10	13	17
1/16	19	23	30	16	19	24
1/32	34	36	44	17	30	36
	Overlap = 1%			Overlap = 2%		
1/4	7	9	11	7	8	10
1/8	10	13	14	10	11	13
1/16	16	19	21	16	17	19
1/32	17	30	32	17	20	30

2.2. One Level Additive. Here are the results. The condition estimate has the same O term as multiplicative, but the text says the iteration counts will be about twice that of the multiplicative. The doubling of the iteration count along the diagonal is pretty clear here.

TABLE 2.3
Iteration Statistics for One-level Additive Schwarz with CG

$H \setminus h$	Overlap = 0				Overlap = 1			
	1/32	1/64	1/128	1/256	1/32	1/64	1/128	1/256
1/4	15	22	30	43	13	17	25	34
1/8	20	28	40	60	15	22	31	46
1/16	26	39	55	80	20	29	42	61
1/32	31	54	77	111	23	40	57	83
	Overlap = 1%				Overlap = 2%			
1/4	13	17	20	25	13	15	18	20
1/8	15	22	26	33	15	18	23	26
1/16	20	29	34	43	20	25	30	34
1/32	23	40	47	58	23	30	41	44

TABLE 2.4
Iteration Statistics for One-level Additive Schwarz with GMRES

$H \setminus h$	Overlap = 0				Overlap = 1			
	1/32	1/64	1/128	1/256	1/32	1/64	1/128	1/256
1/4	15	21	30	42	12	17	23	33
1/8	19	27	39	55	15	21	30	42
1/16	25	36	52	75	20	28	39	57
1/32	29	49	71	103	21	38	53	77
	Overlap = 1%				Overlap = 2%			
1/4	12	17	19	25	12	14	17	19
1/8	15	21	25	31	15	17	22	25
1/16	20	28	33	41	20	23	29	32
1/32	21	38	43	55	21	28	39	42

3. Two Level Methods. The missing part of a two-level method is the coarse mesh. I did this by building a coarse mesh restriction operator R and setting $A_0 = RAR^T$. I stored R as a sparse matrix. R^T is not interpolation, but rather (sort of) piecewise constant approximation. The columns of R , which are 1D arrays, are the 1D translation of basis functions like this. These are built with the matlab function `pu.m`.

FIG. 3.1. *Coarse mesh basis functions*
 $H=1/16, \text{Overlap}=1$ $H=1/16, \text{Overlap}=3$

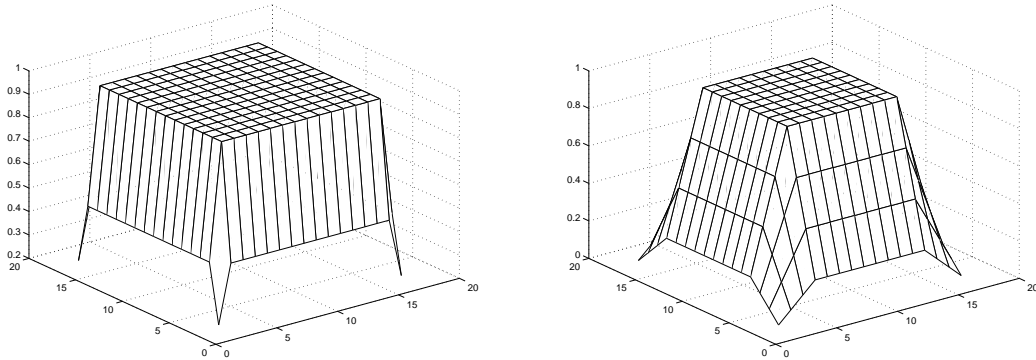


TABLE 3.1
Iteration Statistics for Two-level Additive Schwarz with CG

$H \setminus h$	Overlap = 0				Overlap = 1			
	1/32	1/64	1/128	1/256	1/32	1/64	1/128	1/256
1/4	14	22	31	43	11	16	24	34
1/8	13	18	26	41	9	14	20	27
1/16	8	14	20	31	5	10	14	23
1/32	2	9	14	23		4	11	15
	Overlap = 1%				Overlap = 2%			
1/4	11	16	20	25	11	13	18	20
1/8	9	14	16	20	9	11	14	16
1/16		10	12	15		8	11	12
1/32			9	11			5	8

TABLE 3.2
Iteration Statistics for Two-level Additive Schwarz with GMRES

$H \setminus h$	Overlap = 0				Overlap = 1			
	1/32	1/64	1/128	1/256	1/32	1/64	1/128	1/256
1/4	14	19	27	39	11	15	22	30
1/8	12	17	25	35	9	13	19	27
1/16	8	13	19	27	5	10	14	20
1/32	2	9	14	20		4	10	15
	Overlap = 1%				Overlap = 2%			
1/4	11	15	18	22	11	13	16	18
1/8	9	13	16	20	9	10	13	15
1/16		10	12	15		8	10	12
1/32			8	11			5	8

TABLE 3.3
Iteration Statistics for Two-level Hybrid Schwarz with GMRES

$H \setminus h$	Overlap = 0				Overlap = 1			
	1/32	1/64	1/128	1/256	1/32	1/64	1/128	1/256
1/4	9	12	17	24	8	10	13	19
1/8	7	9	12	17	7	8	11	14
1/16	5	7	9	13	5	7	9	11
1/32	1	5	7	10		4	7	9
	Overlap = 1%				Overlap = 2%			
1/4	8	10	12	14	8	9	11	12
1/8	7	8	9	11	7	7	9	10
1/16		7	7	9		6	7	8
1/32			6	8			5	7

4. Alternating Schwarz. This is different from the other one level methods. This is a two subdomain method. I let $n = 2^p - 1$ and $h = 1/(n + 1)$. On an $n \times n$ mesh, I splint the square with a horizontal line. The two subdomains are both $n \times n/2$ where

$$n/2 = \frac{n + 1}{2} + o - 1.$$

I used the codes from § 1.3 to construct the subdomain matrices, which are not square in this application.

The solver is called `altsw.m`. It keeps track of the differences in successive iterations as a way to estimate the rate of convergence.

```
function [u2d,stats]=altsw(m,rhs2d,uin,overlap)
%
% Alternating Schwarz solver
%
maxit=10;
stats=zeros(maxit,1);
m2=overlap-1+((m+1)/2); h=1/(m+1);
al=pmat2(m,m2,h);
u2d=uin;
%
% Sweep through the subdomains
%
mr=m2;
for i=1:maxit
uold=u2d;
blow=u2d(1:m,mr+1)*h^-2;
rlow=rhs2d(1:m,1:mr);
rlow(1:m,m2)=rlow(1:m,mr)+blow;
rl=rlow(:);
ul=al\rl;
v=zeros(m,m2); v(:)=ul;
u2d(1:m,1:mr)=v;
%
% upper block
%
ml=.5*(m+1)+1-overlap;
bup=u2d(1:m,ml-1)*h^-2;
rup=rhs2d(1:m,ml:m);
rup(1:m,1)=rup(1:m,1)+bup;
rr=rup(:);
ur=al\rr;
v=zeros(m,m2); v(:)=ur;
u2d(1:m,ml:m)=v;
stats(i)= max(max(u2d-uold));
end
```

The alternating Schwarz test program is `alttst.m`. It takes the data from `altsw` and tabulates the iteration statistics for several grids (in this case 3).

```

%
% example of alternating Schwarz iteration
% for  $-u_{xx} - u_{yy} = f$ 
%
%
stats=zeros(10,3);
for p=1:3
mp=p+4;
m=2^mp-1; h=1/(m+1);
x=1:m; x=x'*h; y=x;
%
% overlap = fraction of m means convergence rate is independent of h
%           = constant means convergence rate increases as h decreases
%
overlap=floor(m/10);
%
% Fix the right side so that the solution is
%
%  $u(x,y) = \sin(\pi x) \sin(\pi y) e^x$ 
%
        wx=sin(pi*x); wxp=pi*cos(pi*x); wxpp=-pi*pi*wx;
        wy=wx; wypp=wxpp;
        u=(wx.*exp(x))*wy';
        wxx=(wx + 2*wxp + wxpp).*exp(x); uxx=wxx*wy';
        wyy=wypp; uyy=(wx.*exp(x))*wyy';
        rhs2d=-(uxx+uyy);
%
%
%
uin=zeros(m,m);
[u2d,stats(:,p)]=altsw(m,rhs2d,uin,overlap);
u1=u2d(:); r1=rhs2d(:); err=norm(lapmf(u1)-r1,inf)
end
%
% look at the convergence reates
%
stats(2:10,:)./stats(1:9,:)

```

As for results. If the overlap is roughly 10%, we see a convergence rate of $\approx .27$ independently of h . This rate sets in pretty early and we only need to examine 5 iterations to see it. For an overlap of three, things are significantly worse and you can see the rate deplete as the mesh is refined. In the table we tabulate the ratios of successive steps in the linear iteration in each column for different values of h and overlap.

TABLE 4.1
Alternating Schwarz Convergence Rates

k \ h	Overlap = 10%			Overlap = 3		
	1/32	1/64	1/128	1/32	1/64	1/128
1	0.51	0.51	0.51	0.51	0.62	0.53
2	0.27	0.27	0.27	0.27	0.51	0.71
3	0.27	0.27	0.27	0.27	0.52	0.71
4	0.28	0.27	0.27	0.28	0.52	0.72
5	0.28	0.27	0.27	0.28	0.52	0.72