Numerical methods, Fall 2018
Final Project:
Developing a Multigrid solver

Ghanesh Narasimhan

January 14, 2019

Contents

1 Problem definition

The 2-D Laplace equation

$$u_{xx} + u_{yy} = 0, \tag{1}$$

is condsidered on $0 \leq x, y \leq 2\pi$ with boundary conditions,

$$u(0, y) = \sin(4y), u(2\pi, y) = 0, u(x, 2\pi) = 0, u(x, 0) = \sin(4x).$$

A numerical solution of this equation should be obtained using a multigrid solver implemented with different relaxation schemes such as:

1. Point Gauss-Seidel iterative scheme,
2. Alternating Directional line Gauss-Seidel method.

2 A discussion on smoothers or relaxation schemes and multigrid method

Consider the 2D Laplace equation (1),

$$u_{xx} + u_{yy} = 0,$$

on $0 \leq x \leq 2\pi$. Three different grid sizes are considered: $256 \times 256,\ 128 \times 128,\ 64 \times 64$. The 2D Laplace equation is discretized using the following second order central difference scheme:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = R_{i,j}, \tag{2}$$

where R is the source term of the elliptic equation and in this case, $R = 0$. This central difference scheme is solved iteratively using point Gauss-Seidel and Line Gauss Seidel relaxation methods.

1. **Point Gauss-Seidel (GS) method** In this method, the solution procedure is similar to Point-Jacobi method except that the updated value is used immediately in the next iteration to accelerate the convergence. From (2), the following scheme is obtained:

$$u_{i,j}^{n+1} = \frac{R_{i,j}}{b} - \frac{a}{b}(u_{i-1,j}^{n+1} + u_{i+1,j}^{n}) - \frac{c}{b}(u_{i,j-1}^{n+1} + u_{i,j+1}^{n}), . \tag{3}$$

where $a = \dfrac{1}{\Delta x^2}$, $b = -2\left(\dfrac{1}{\Delta x^2} + \dfrac{1}{\Delta y^2}\right)$, $c = \dfrac{1}{\Delta y^2}$. It is evident from (3) that the updated values at $(i-1, j), (i, j-1)$ are immediately used to update the value at (i, j).

The point GS method can be implemented with a successive over relaxation parameter. In this scheme, firstly an update value (u^*) is obtained using Gauss-Seidel method (3). Then, a parameter ω is chosen such that $0 < \omega < 2$ and the solution at $n+1$ iteration is obtained as

$$u^{n+1} = (1-\omega)u^n + \omega u^*. \tag{4}$$

When $\omega < 1$, it is called under relaxation and when $\omega > 1$, it is called as over relaxation.

2. **Line Gauss-Seidel in alternating directions (Line-GS)**

Consider the equation (2)

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = R_{i,j}.$$

Now, this equation is evaluated as a two step procedure where firstly the solution is obtained for $n+1/2$ iteration and then the solution is obtained for $n+1$ iteration as follows

$$\begin{aligned} &\text{(i) } a\ u_{i+1,j}^{n+1/2} + b\ u_{i,j}^{n+1/2} + a\ u_{i-1,j}^{n+1/2} = R_{i,j} - c\ (u_{i,j+1}^{n} + u_{i,j-1}^{n}),\\ &\text{(ii) } \quad c\ u_{i,j+1}^{n+1} + b\ u_{i,j}^{n+1} + c\ u_{i,j-1}^{n+1} = R_{i,j} - a\ (u_{i+1,j}^{n+1/2} + u_{i-1,j}^{n+1/2}), \end{aligned}$$

where $a = \dfrac{1}{\Delta x^2}$, $b = -2\left(\dfrac{1}{\Delta x^2} + \dfrac{1}{\Delta y^2}\right)$, $c = \dfrac{1}{\Delta y^2}$. These equations constitute the alternating directional Gauss Seidel method. It is evident here that in step (i), the scheme is implicit along x direction and in step (ii), the y direction is implicit. Similar to (4) scheme, an update value u^* can be obtained from the Line GS and u^{n+1} is found from

$$u^{n+1} = (1-\omega)u^n + \omega u^*,$$

where ω is the over-relaxation parameter.

3. **Multigrid method**

Consider the Laplace equation (1),

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = R,$$

where $R = 0$ in the current problem. Let $L = \dfrac{\partial^2}{\partial x^2} + \dfrac{\partial^2}{\partial y^2}$ be the Laplace operator operating linearly on u such that (1) becomes

$$L(u) = R \tag{5}$$

The idea of multigrid method is that in a given grid, high frequency errors decay faster than low frequency error. Using this idea, the elliptic equaiton is solved recursively by successive coarsening of the grid such that the low frequency errors on a fine grid would appear as high frequency components and thereby decays faster on a coarser grid. The process of solving equations using multigrid method involves:

(a) **Restriction**
Consider a grid with $N_x \times N_y$ points. Let the solution obtained in this grid be u_f. This solution would satisfy

$$L(u_f) = R + \epsilon,$$

where ϵ is the error associated with the fine grid. Since, $L(u) = R$,

$$\begin{aligned} L(u_f) &= L(u) + \epsilon, \\ L(u - u_f) &= -\epsilon, \\ L(u_\epsilon) &= -\epsilon, \end{aligned} \tag{6}$$

Now, at this stage, suppose the solution for u_ϵ is obtained by solving (6), then the solution u can be obtained by

$$\begin{aligned} L(u_f) &= R - L(u_\epsilon), \\ L(u_f + u_\epsilon) &= L(u), \\ \implies u &= u_f + u_\epsilon. \end{aligned} \tag{7}$$

However, instead of solving for u_ϵ on the finer grid, (6) is solved iteratively on a coarser grid for faster convergence. This process of solving (6) on a coarser grid is called *Restriction.*

(b) **Prolongation**
The u_ϵ obtained on the coarser grid is interpolated back to the finer grid and added back to the older fine grid solution u_f as in (7). This process of interpolation of coarse grid solution is called *Prolongation.*

This process of restriction and prolongation are combined in different ways and different multigrid methods are obtained such as:

(a) V-Cycle
(b) W-Cycle
(c) Full Multigrid (FMG) Cycle.

In this project, the V-Cycle method is implmented and the performance of the solver is discussed using three different grids: $256 \times 256, 128 \times 128, 64 \times 64$. A comparison between V cycle and W cycle is also reported. The convergence of the multigrid solver is studied by computing the residue at each cycle. The residue is defined the same way as in (6),

$$\epsilon = \|\nabla^2 u - R\|, \tag{8}$$

where $L = \nabla^2 = \dfrac{\partial^2}{\partial x^2} + \dfrac{\partial^2}{\partial y^2}$.

3 (b) Flow-chart of the solver

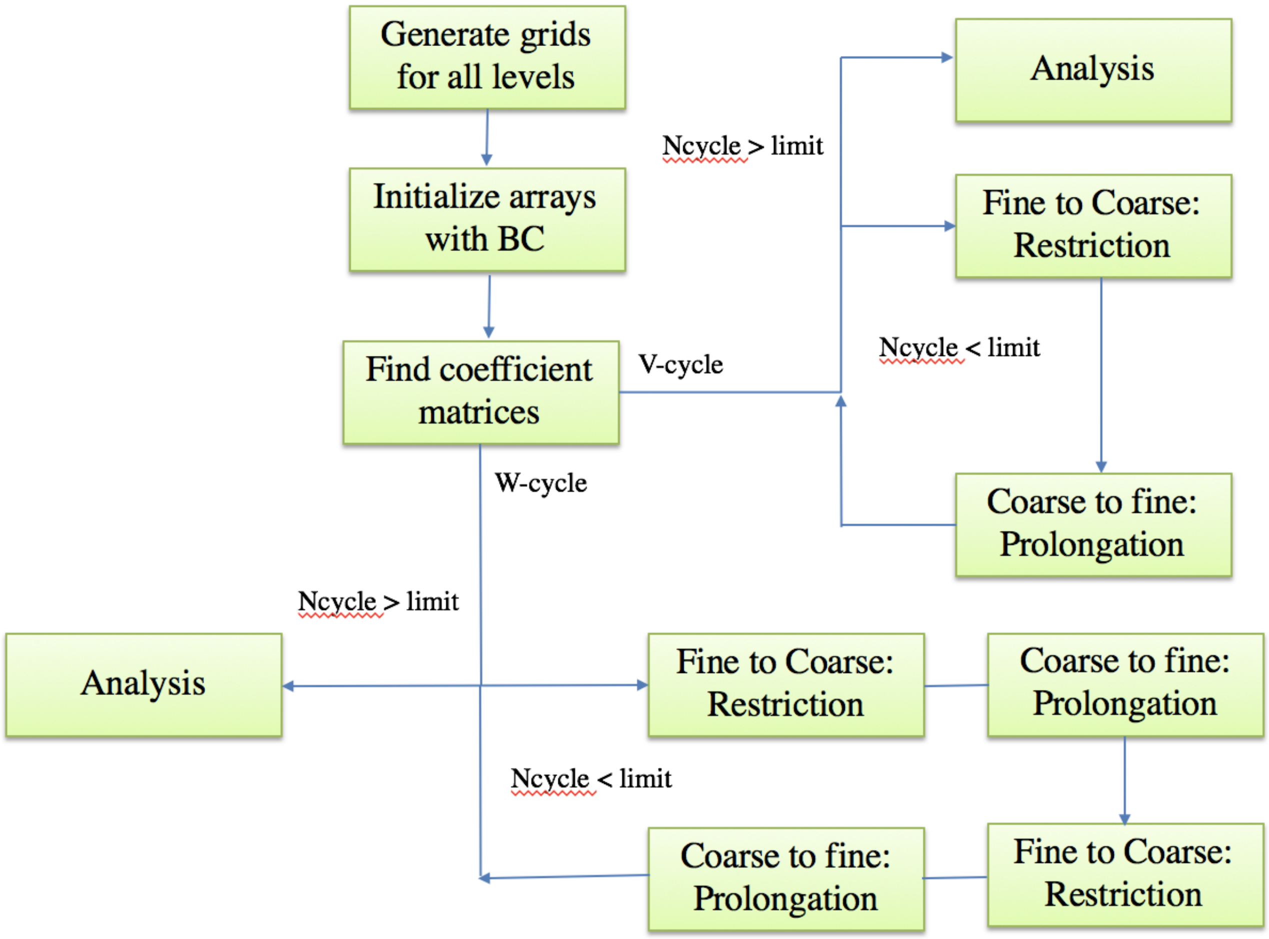
Generate grids for all levels
Initialize arrays with BC
Find coefficient matrices
V-cycle
Ncycle > limit
Analysis
Fine to Coarse: Restriction
Ncycle < limit
Coarse to fine: Prolongation
W-cycle
Ncycle > limit
Analysis
Fine to Coarse: Restriction
Coarse to fine: Prolongation
Ncycle < limit
Fine to Coarse: Restriction
Coarse to fine: Prolongation

Figure 1: Flow chart of the multigrid solver.

4 Results and discussion

(c) Solutions on different grid

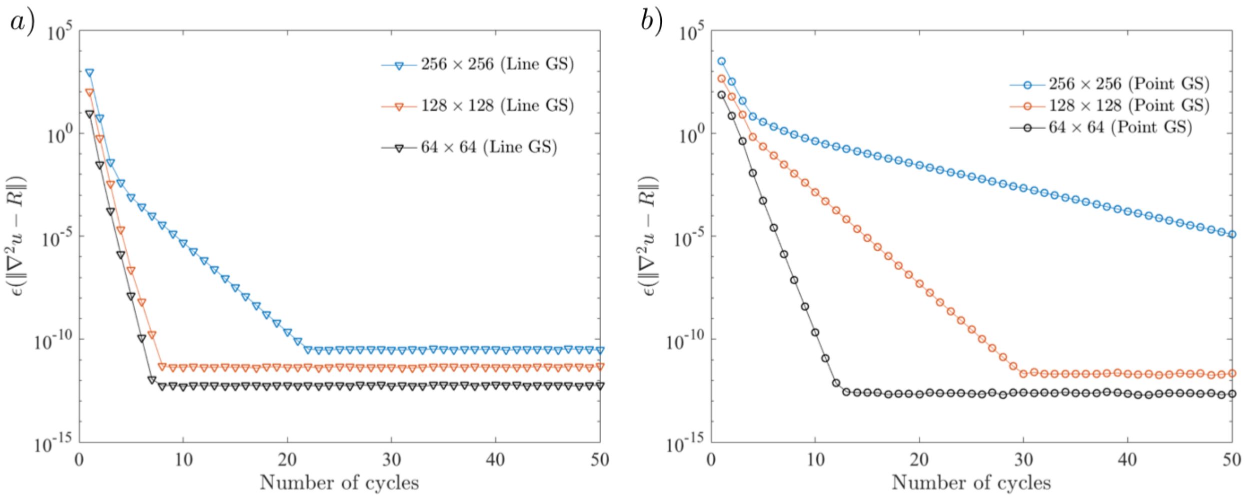

Figure 2: ϵ plotted against number of V cycles for (a) Line Gauss Seidel and (b) Point Gauss Seidel smoothers on different grids $256 \times 256, 128 \times 128, 64 \times 64$.

The Laplace equation is solved on different initial grids $256 \times 256, 128 \times 128, 64 \times 64$. The residue ϵ defined in (8) is plotted against number of V cycles using Line GS and Point GS smoothers. A total of 5 levels and 5 maximum iterations is used for all the three different initial grid sizes. Within one iteration in Line GS, the grid is swept through twice owing to Alternating Direction method. This essentially means that Line GS has 10 meximum iterations per level.

Plot 2(a) and 2(b) shows the convergence behaviour of multigrid solver for the three grids. The number of cycles required for 256×256 is higher than the coarser grids as high frequency errors damp out quickly on coarser grids. Both Line GS and point GS smoothers reproduce this trend where the finest grid takes more cycles to converge than the coarsest grid.

4.1 (d) Sanity check

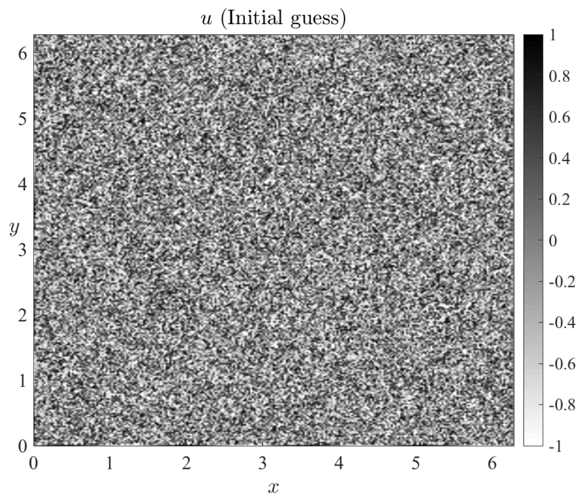

Figure 3: Initial guess generated from random numbers between $[-1, 1]$.

The multigrid solver gives solution to the Laplace equation iteratively starting from a random initial guess as shown in 3. It is expected that the multigrid solver with Line GS would converge faster than the point GS method as Line GS solves for many points at a time. This faster convergence of Line GS method is shown in figure 4. For the same total number of iterations (2500) across 50 V cycles, point GS converges to $\epsilon \sim 10^{-10}$ around 45 V cycles whereas Line GS reaches this limit within 21 cycles. This shows that the iterative solvers has proper convergence properties.

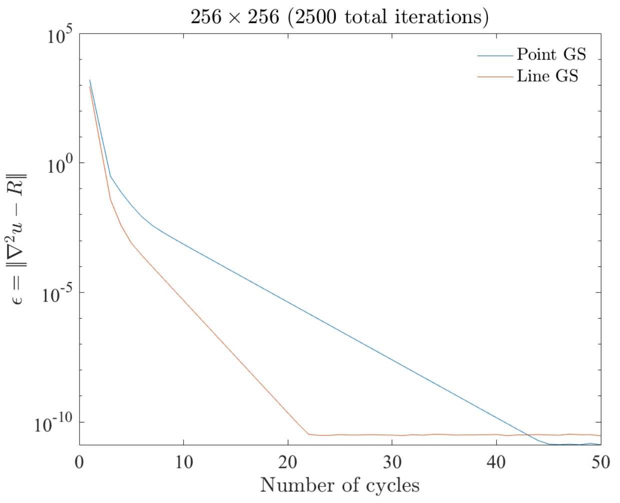

Figure 4: $\epsilon = \|\nabla^2 u - R\|$ plotted against number of V cycles on a 256×256 grid using Point GS and Line GS with a fixed number of total iterations (2500).

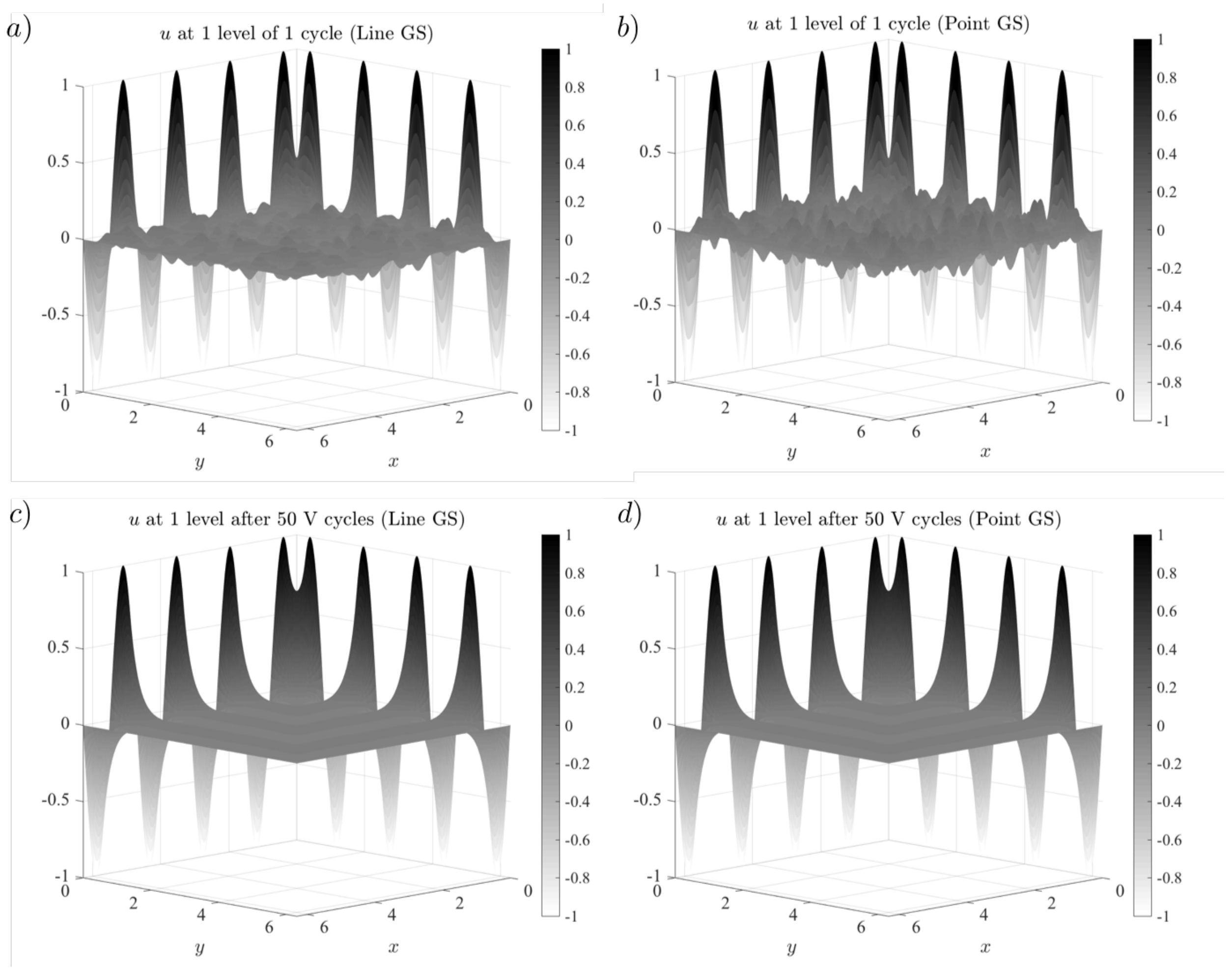

Figure 5: u solution contours at level 1 of 1st and 50th V cycle for Line GS (a, c) and Point GS (b, d).

Figure 5 shows the initial and final u solutions after 50 V cycles (2500 total iterations). It is evident that the errors owing to the random initial guess is quite significant in figures $5(a)$ and $5(b)$ using Line and point GS iterative solvers respectively. The high frequency erros in the first V cycle has been completely damped in the contours plotted at the 50th cycle (figures $5(c)$ and $5(d)$). The boundary conditions are also satisfied throughout the V cycles. This suggests that the multigrid solver is able to solve for the right solutions. Therefore, from the figures of (5) and (4), the multigrid solver implmented for this project has the right convergence behaviour. Further tests are performed where the convergence of the solver is studied by varying the number of coarse grid levels for the finest grid. Also, the CPU time is also computed for solving the Laplace equation with and without the multigrid solver.

(e) (i) Convergence with number of coarse grid levels

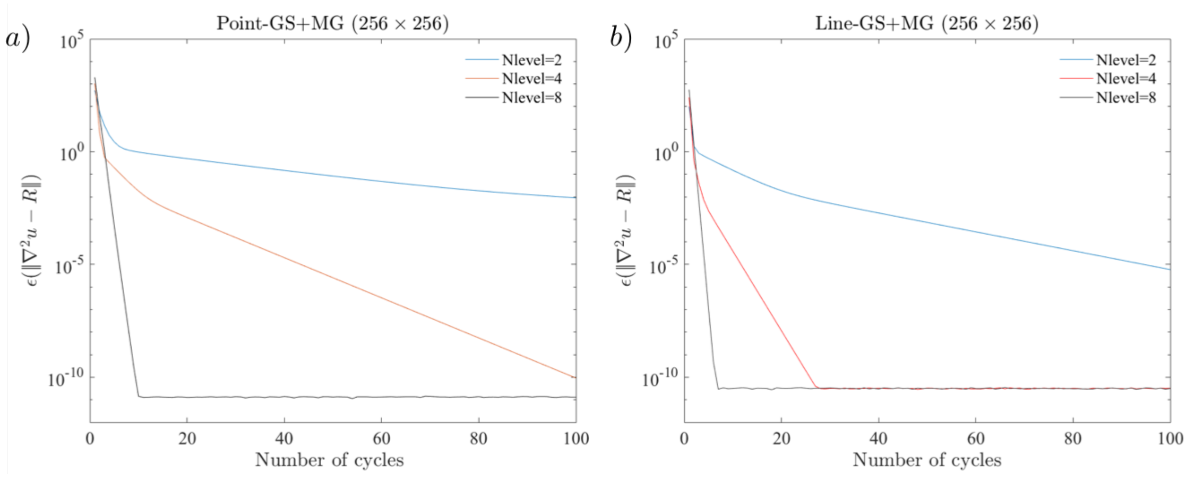

Figure 6: ϵ against number of V cycles for different coarse grid levels using (a) point GS and (b) Line GS smoothers.

In figure (6), the ϵ dependence on the number of coarse grid levels is studied. These curves are obtained for both Line and point GS relaxation methods on a 256×256 grid. Residue is obtained across 100 V cycles using 2,4 and 8 coarse grid levels. To keep the total number of iterations (6400 in this case) constant, the maximum iterations per level for the smoothers is varied accordingly to the number of coarse grid levels.

As the number of levels is increased, the high frequency errors would decay faster. This means that the solver with large number of coarse grid levels would converge quickly. This trend is reproduced by both Line and point GS as shown in figure (6). With 8 coarse grid levels, both iterative relaxation schemes is able to converge within 10 V cycles. With 4 levels, Line GS converges around 30 V cycle and point GS would converge after 100 V cycles. It would take several V cycles if one starts with 2 levels.

(e) (ii) CPU time with and without multigrid

Iterative solver	CPU time (s) to reach $\epsilon = 10^{-5}$	Total iterations
Point GS (Non-MG)	2500	60708
Line GS (Non-MG)	1222.6	15182
Point GS (MG)	2.36	950 (19 V cycles)
Line GS (MG)	3.46	500 (10 V cycles)

Table 1: CPU time comparison with and without multigrid.

The CPU time is computed for the non-multigrid and multigrid sovlers. The results are summarised in table 1. The time taken to reach the residual value $\epsilon \sim 10^{-5}$ is obtained for point and Line

GS solvers with and without the multigrid procedure. The 256 × 256 grid is used to obtain these values in the table. The multigrid solvers use 5 coarse grid levels with 10 maximum iterations at each level.

The point GS method without multigrid takes around 40 minutes CPU time to reach $\epsilon \sim 10^{-5}$ in 60708 iterations. Line GS without multigrid takes around 20 minutes to reach the same residual limit in 15182 iterations. Whereas, it just takes 2-3 seconds to reach the same order of accuracy in less number of iterations when multigrid is employed. Point GS converged within 950 iterations (19 cycles) and Line GS converged in 500 iterations (10 V cycles).

(g)(iii) Convergence properties of W cycle

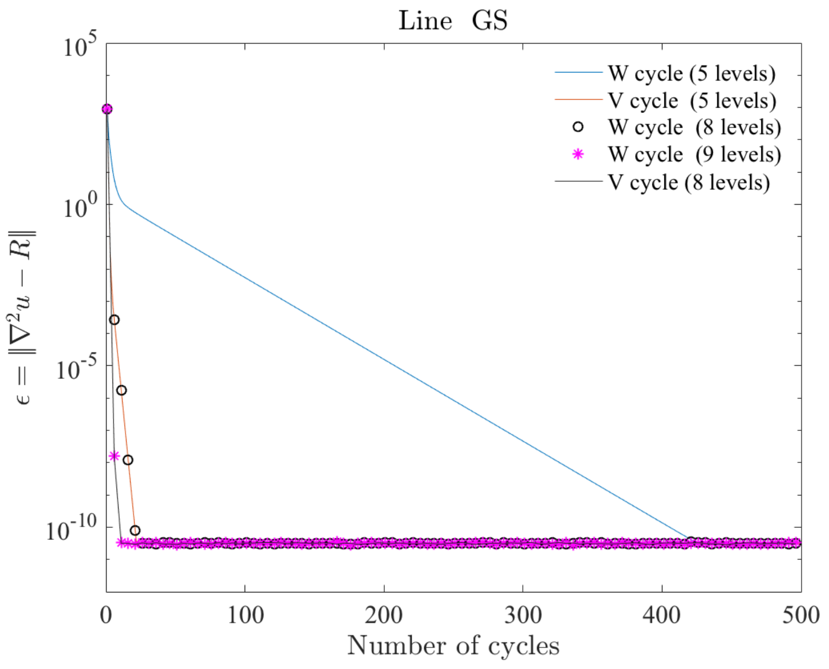

Figure 7: ϵ against number of W cycles for different coarse grid levels using Line Gauss Seidel.

Figure (7) shows the convergence properties of the W cycle. A comparison with the V cycle is also made. The W cycles are computed on a 256 × 256 grid. The W cycle goes as $[1, 2, 3, 4, 5, 4, 5, 4, 3, 4, 5, 4, 3, 2, 1]$. A maximum of 10 Line GS iterations is performed. The convergence is tested by varying number of coarse grid levels.

The figure 7 shows that W cycle converges slower than V cycle when 5 grid levels are considered. It takes around 400 W cycles on a 256 × 256 grid. Correspondingly, V cycle just takes 8 cycles to reach convergence with 5 levels. This slower convergence of W cycle could be attributed to introducing new interpolation errors at the intermediate prolongation stages. It takes more number of cycles for the W cycle to damp these additional errors.

These errors in a W cycle can be damped faster if the to and fro branch of W cycle is performed at the coarsest grid levels. This is shown in the figure. When 8 grid levels are considered, W cycle is as fast as the V cycle with 5 levels. W cycle with 9 levels is faster than V cycle with 5 levels and is as fast as V cycle with 8 levels. This observation shows that the W branch if taken to the coarsest grid levels would accelerate convergence and would be significantly slower if the W branch operates on a relatively finer grid.

5 Conclusion

The Laplace equation is solved using the multigrid method with Line and point Gauss Seidel schemes as smoothers. Solutions are obtained in three different initial grids. It is observed that the coarser grids converge faster. This is because the high frequency errors decay much faster in coarser grid. For the same total number of iterations, point GS converges slower than line GS. Upon increasing the number of coarse grid levels, the solutions converged faster as much coarser grids are available. The CPU time significantly decreased from several minutes to few seconds when multigrid method is used. All these observations points that the solver works correct with proper convergence trends. When W cycles are used, it's observed that for the same multigrid parameters, V cycle converges faster. Increase in the number of coarse grid levels accelerated the convergence in W cycle.

Appendices

Contents

```
%%**************MULTIGIRD PROJECT******************
%%*****SUBMITTED BY GHANESH NARASIMHAN*************
%%*******NUMERICAL METHODS (FALL 2018)*************

clear
```

Multigrid parameters

```
Nlvlmax=5;            %Number of multigrid levels
Ncycle=50;           %Number of MG cycles
%tot_iter=2500;       %total iterations=Ncycle*maxiter*Nlvlmax;
maxiter=5;%tot_iter/(Nlvlmax*Ncycle);%Maximum iterations in iterative solver

omg=1.00;            %SOR parameter
Vcycle=1;            %Flag for V-Cycle
Wcycle=0;            %Flag for W-Cycle


%Type of smoother
itype=3;
%
%itype=1 Point Gauss-Seidel red black + SOR + MG
%itype=2 Point Gauss-Seidel normal    + SOR + MG
%itype=3 Line  Gauss-Seidel (ADI)     + SOR + MG
%itype=4 Line  Gauss-Seidel (ADI)     + SOR + Non-MG
%itype=5 Point Gauss-Seidel           + SOR + Non-MG
```

Grid definition for each level

```
Lx=2*pi;Ly=2*pi;
Nx=257;Ny=257;
nx(1)=Nx;
ny(1)=Ny;

%Number of grid points at other levels
for i=2:Nlvlmax
ny(i)=((ny(1)-1)/2^(i-1))+1;
nx(i)=((nx(1)-1)/2^(i-1))+1;
end
```

Initialise arrays on each level

```
error(1:Ncycle)=0;
for i=1:Nlvlmax
uin{i}(1:ny(i),1:nx(i))=0;
uout{i}(1:ny(i),1:nx(i))=0;
uoutnew{i}(1:ny(i),1:nx(i))=0;
eps{i}(1:ny(i),1:nx(i))=0;
epsnew{i}(1:ny(i),1:nx(i))=0;
RHS{i}(1:ny(i),1:nx(i))=0;
end
```

Initial condition (random noise+ boundary condition)

```
for n=1:Nlvlmax
dx=Lx/(nx(n)-1);dy=Ly/(ny(n)-1);
x=0:dx:Lx;
y=0:dy:Ly;
if (n==1)
%uin{n}= -1 + (1+1)*rand(ny(1),nx(1));
u=load('init_cond.mat');
uin{n}(1:ny(n),1:nx(n))=u.u(1:ny(n),1:nx(n));
uin{n}(1:ny(n),1)=sin(4*y);
uin{n}(1:ny(n),nx(n))=0;
uin{n}(1,1:nx(n))=sin(4*x);
uin{n}(ny(n),1:nx(n))=0;
end
end
```

Calculate co-efficients

```
a(1:Nlvlmax)=0;b(1:Nlvlmax)=0;c(1:Nlvlmax)=0;
for n=1:Nlvlmax
A{n}=0;B{n}=0;invA{n}=0;invB{n}=0;
```

```
[invA{n},invB{n},A{n},B{n},a(n),b(n),c(n)]=coeff(n,Nx,Ny,Lx,Ly);
end
```

Solving Lu=R using Non-MG

```
if (itype>=4)
n=1;tol=1e-5;counternMG=0;
[uout{n},counternMG,tnonMG,errornMG]=NonMG(uin{n},RHS{n},invA{n},invB{n}...
,a(n),b(n),c(n),Nx,Ny,itype,omg,tol);
semilogy(errornMG)
end
```

Solving Lu=R using MG (V-cycle)

```
if (Vcycle==1 && itype<4)
tsV=cputime;
for Ncyl=1:Ncycle
Ncyl
[uout]=fine_to_coarse(uin,RHS,maxiter,invA,invB,A,B,a,b,c,...
nx,ny,itype,omg,Nlvlmax,1,uout,eps,epsnew);
[uout]=coarse_to_fine(Nlvlmax,uout,uoutnew,2);
uin{1}=uout{1};
%residue for testing convergence
error(Ncyl)=norm(residual(uout{1},RHS{1},a(1),b(1),c(1),nx(1),ny(1)));
end
end
```

Solving Lu=R using MG (W-cycle)

```
if (Wcycle==1 && itype<4 && Nlvlmax>3)
tsW=cputime;
for Ncyl=1:Ncycle
Ncyl
[uout]=fine_to_coarse(uin,RHS,maxiter,invA,invB,A,B,a,b,c,nx,ny,itype,...
omg,Nlvlmax,1,uout,eps,epsnew);
[uout]=coarse_to_fine(Nlvlmax,uout,uoutnew,Nlvlmax-1);
uin{Nlvlmax-2}=uout{Nlvlmax-2};
[uout]=fine_to_coarse(uin,RHS,maxiter,invA,invB,A,B,a,b,c,nx,ny,itype,omg,...
Nlvlmax,Nlvlmax-2,uout,eps,epsnew);
[uout]=coarse_to_fine(Nlvlmax,uout,uoutnew,2);
uin{1}=uout{1};
%residue for testing convergence
error(Ncyl)=norm(residual(uout{1},RHS{1},a(1),b(1),c(1),nx(1),ny(1)));
end
teW=cputime-tsW;
end
```

Analysis

```
figure(3)
Lx=2*pi;Ly=2*pi;
dx=Lx/(nx(1)-1);dy=Ly/(ny(1)-1);
x=0:dx:2*pi;
y=0:dy:2*pi;
[X,Y]=meshgrid(x,y);
surf(x,y,uout{1},'linestyle','none')
set(gca, 'CameraPosition', [2*pi 2*pi 0.25]);
xlabel('$x$','interpreter','latex','fontsize',16)
ylabel('$y$','interpreter','latex','fontsize',16)
title('$u$ (Line GS)','interpreter','latex','fontsize',16)
set(gcf,'Color','w')
set(gca,'fontsize',16,'fontname','times')
colorbar
colormap(flipud(gray))

figure(1)
semilogy(error);hold on
xlabel('Number of cycles','interpreter','latex','fontsize',16)
ylabel('$\epsilon =|\!|\nabla^2u-R|\!|$','interpreter','latex','fontsize',16)
title('Point \ GS','interpreter','latex','fontsize',16)
set(gcf,'Color','w')
set(gca,'fontsize',16,'fontname','times')
```

Subroutines

Iterative Solve

1.
```
function [uout]=iterative_solve(uin,RHS,maxiter,...
invA,invB,A,B,a,b,c,Nx,Ny,itype,omg)
```

Point Gauss-Seidel + MG (red-black+SOR)

```
if (itype==1)
uout(1:Ny,1:Nx)=0;
utemp(1:Ny,1:Nx,1:2)=0;
utemp(1:Ny,1:Nx,1)=uin(1:Ny,1:Nx);
utemp(1,:,2)=utemp(1,:,1);
utemp(Ny,:,2)=utemp(Ny,:,1);
utemp(:,1,2)=utemp(:,1,1);
utemp(:,Nx,2)=utemp(:,Nx,1);

for iter=1:maxiter
for i=2:Nx-1
for j=2:Ny-1
if (mod(i+j,2)~=0)
utemp(j,i,2)=RHS(j,i)*(1/b)-(a/b)*(utemp(j,i-1,1)+utemp(j,i+1,1))...
-(c/b)*(utemp(j-1,i,1)+utemp(j+1,i,1));
end
end
end

for i=2:Nx-1
for j=2:Ny-1
if (mod(i+j,2)==0)
utemp(j,i,2)=RHS(j,i)*(1/b)-(a/b)*(utemp(j,i-1,2)+utemp(j,i+1,2))...
-(c/b)*(utemp(j-1,i,2)+utemp(j+1,i,2));
end
end
end
%SOR
utemp(2:Ny-1,2:Nx-1,2)=utemp(2:Ny-1,2:Nx-1,2)*omg+...
utemp(2:Ny-1,2:Nx-1,2)*(1-omg);
utemp(1:Ny,1:Nx,1)=utemp(1:Ny,1:Nx,2);
end
uout(1:Ny,1:Nx)=utemp(1:Ny,1:Nx,2);%utemp copied with boundary values
end
```

Point Gauss-Seidel + MG (normal)

```
if (itype==2)
utemp(1:Ny,1:Nx,1:2)=0;
utemp(1:Ny,1:Nx,1  )=uin(1:Ny,1:Nx);
utemp(1   ,:   ,2  )=utemp(1 ,: ,1);
utemp(Ny  ,:   ,2  )=utemp(Ny,: ,1);
utemp(:   ,1   ,2  )=utemp(: ,1 ,1);
utemp(:   ,Nx  ,2  )=utemp(: ,Nx,1);
uout(1:Ny,1:Nx     )=0;
for iter=1:maxiter
for i=2:Nx-1
for j=2:Ny-1
utemp(j,i,2)=RHS(j,i)*(1/b)-(a/b)*(utemp(j,i-1,2)+utemp(j,i+1,1))...
-(c/b)*(utemp(j-1,i,2)+utemp(j+1,i,1));
end
end
%SOR
utemp(2:Ny-1,2:Nx-1,2)=utemp(2:Ny-1,2:Nx-1,2)*omg...
+utemp(2:Ny-1,2:Nx-1,2)*(1-omg);
```

```
utemp(1:Ny,1:Nx,1)=utemp(1:Ny,1:Nx,2);
end
uout(1:Ny,1:Nx)=utemp(1:Ny,1:Nx,2);%utemp copied with boundary values

end
```

Line-SOR ADI + MG

```
if (itype==3)
utemp(1:Ny,1:Nx,1:3)=0;
utemp(1:Ny,1:Nx,1  )=uin(1:Ny,1:Nx);
utemp(1   ,:   ,2  )=utemp(1 ,: ,1);
utemp(Ny  ,:   ,2  )=utemp(Ny,: ,1);
utemp(:   ,1   ,2  )=utemp(: ,1 ,1);
utemp(:   ,Nx  ,2  )=utemp(: ,Nx,1);

utemp(1   ,:   ,3  )=utemp(1 ,: ,1);
utemp(Ny  ,:   ,3  )=utemp(Ny,: ,1);
utemp(:   ,1   ,3  )=utemp(: ,1 ,1);
utemp(:   ,Nx  ,3  )=utemp(: ,Nx,1);

for iter=1:maxiter
for j=2:Ny-1
BCmatx(1,1:Nx-2)=0;
BCmatx(1,   1)=-a*utemp(j ,1 ,1);
BCmatx(1,Nx-2)=-a*utemp(j ,Nx,1);
Atemp(1:Nx-2,1)=RHS(j,2:Nx-1)-c*(utemp(j+1,2:Nx-1,1)...
+utemp(j-1,2:Nx-1,2))+BCmatx(1,1:Nx-2);
utemp(j,2:Nx-1,2)=mtimes(invB,Atemp);
end

for i=2:Nx-1
BCmaty(1:Ny-2,1)=0;
BCmaty(1,   1)=-c*utemp(1 ,i,1);
BCmaty(Ny-2,1)=-c*utemp(Ny,i,1);
Btemp(1:Ny-2,1)=RHS(2:Ny-1,i)-a*(utemp(2:Ny-1,i-1,3)...
+utemp(2:Ny-1,i+1,2))+BCmaty(1:Ny-2,1);
utemp(2:Ny-1,i,3)=mtimes(invA,RHS(2:Ny-1,i)...
-a*(utemp(2:Ny-1,i-1,3)+utemp(2:Ny-1,i+1,2))+BCmaty(1:Ny-2,1));

%SOR
utemp(2:Ny-1,2:Nx-1,3)=utemp(2:Ny-1,2:Nx-1,1)*(1-omg)+utemp(2:Ny-1,2:Nx-1,3)*omg;
utemp(1:Ny,1:Nx,1)=utemp(1:Ny,1:Nx,3);
end
```

```
uout(1:Ny,1:Nx)=utemp(1:Ny,1:Nx,3);...

%utemp copied with boundary values
end
end
```

Fine to coarse

2.
```
function [uout]=fine_to_coarse(uin,RHS,maxiter,invA,invB,A,B,a,b,c,nx,ny,...
itype,omg,Nlvlmax,Nlvlmin,uout,eps,epsnew)

%Fine to coarse
for n=Nlvlmin:Nlvlmax
fprintf('na=%d \n',n)
%Solve for Lu=R iteratively "maxiter" times
uout{n}=iterative_solve(uin{n},RHS{n},maxiter,invA{n},...
invB{n},A{n},B{n},a(n),b(n),c(n),nx(n),ny(n),itype,omg);
%Compute the residual from the smoothened solution
eps{n}=residual(uout{n},RHS{n},a(n),b(n),c(n),nx(n),ny(n));
%Define the new RHS for the next level
if(n~=Nlvlmax)
epsnew{n+1}=restriction(eps{n},nx(n),ny(n));
RHS{n+1}=-epsnew{n+1};
uin{n+1}(1:ny(n+1),1:nx(n+1))=0;
end
end


end
```

3. **Coarse to fine**

```
function [uout]=coarse_to_fine(Nlvlmax,uout,uoutnew,Nlvlmin)
%Coarse to fine
for n=Nlvlmax:-1:Nlvlmin
fprintf('nb=%d \n',n-1)
uoutnew{n-1}=prolongation(uout{n});
uout{n-1}=uout{n-1}+uoutnew{n-1};
end


end
```

Restriction

4.
```
function [epsnew]=restriction(eps,Nx,Ny)
Nxnew=(Nx-1)/2+1;
Nynew=(Ny-1)/2+1;

epsnew(1:Nynew,1:Nxnew)=0;
%Perform restriction by copying every alternate points in the grid
epsnew(1:Nynew,1:Nxnew)=eps(1:2:Ny,1:2:Nx);
end
```

5. **Prolongation**

```
function [uoutnew]=prolongation(uout)
[Ny,Nx]=size(uout);
Nxnew=(Nx-1)*2+1;
Nynew=(Ny-1)*2+1;

%Prolongation step
uoutnew(1:Nynew,1:Nxnew)=0;
uoutnew(1:Nynew,1:Nxnew)=interp2(uout);


end
```

Residual

6.
```
function [eps]=residual(uout,RHS,a,b,c,Nx,Ny)
eps(1:Ny,1:Nx)=0;
%Compute residual \epsilon=Lu-R
for i=2:Nx-1
for j=2:Ny-1
eps(j,i)=a*(uout(j,i-1)+uout(j,i+1))+...
c*(uout(j-1,i)+uout(j+1,i))+b*uout(j,i)-RHS(j,i);
end
end

end
```

7. **Non-Multigrid iterative solvers**

Line-SOR ADI (non-MG)

```
if (itype==4)
```

```
utemp(1:Ny,1:Nx,1:3)=0;
utemp(1:Ny,1:Nx,1  )=uin(1:Ny,1:Nx);
utemp(1   ,:   ,2  )=utemp(1 ,: ,1);
utemp(Ny  ,:   ,2  )=utemp(Ny,: ,1);
utemp(:   ,1   ,2  )=utemp(: ,1 ,1);
utemp(:   ,Nx  ,2  )=utemp(: ,Nx,1);

utemp(1   ,:   ,3  )=utemp(1 ,: ,1);
utemp(Ny  ,:   ,3  )=utemp(Ny,: ,1);
utemp(:   ,1   ,3  )=utemp(: ,1 ,1);
utemp(:   ,Nx  ,3  )=utemp(: ,Nx,1);
error=1;
counter=0;
ts=cputime;
while(error>tol)
counter=counter+1
for j=2:Ny-1
BCmatx(1,1:Nx-2)=0;
BCmatx(1,   1)=-a*utemp(j ,1 ,1);
BCmatx(1,Nx-2)=-a*utemp(j ,Nx,1);
Atemp(1:Nx-2,1)=RHS(j,2:Nx-1)-c*(utemp(j+1,2:Nx-1,1)...
+utemp(j-1,2:Nx-1,2))+BCmatx(1,1:Nx-2);
utemp(j,2:Nx-1,2)=mtimes(invB,Atemp);
end

for i=2:Nx-1
BCmaty(1:Ny-2,1)=0;
BCmaty(1,   1)=-c*utemp(1 ,i,1);
BCmaty(Ny-2,1)=-c*utemp(Ny,i,1);
Btemp(1:Ny-2,1)=RHS(2:Ny-1,i)-a*(utemp(2:Ny-1,i-1,3)...
+utemp(2:Ny-1,i+1,2))+BCmaty(1:Ny-2,1);
utemp(2:Ny-1,i,3)=mtimes(invA,Btemp);
end
%SOR
utemp(2:Ny-1,2:Nx-1,3)=  utemp(2:Ny-1,2:Nx-1,1)*(1-omg)...
+utemp(2:Ny-1,2:Nx-1,3)*omg;
%Stopping criteria
error=norm(residual(utemp(1:Ny,1:Nx,3),RHS,a,b,c,Nx,Ny))
errora(counter)=error;
if (error>tol)
utemp(2:Ny-1,2:Nx-1,1)=utemp(2:Ny-1,2:Nx-1,3);
else
uout(1:Ny,1:Nx)=utemp(1:Ny,1:Nx,3);
end
end
```

```
te=cputime-ts;

end
```

Point GS+SOR

```
if (itype==5)

utemp(1:Ny,1:Nx,1:2)=0;
utemp(1:Ny,1:Nx,1  )=uin(1:Ny,1:Nx);
utemp(1   ,:   ,2  )=utemp(1 ,: ,1);
utemp(Ny  ,:   ,2  )=utemp(Ny,: ,1);
utemp(:   ,1   ,2  )=utemp(: ,1 ,1);
utemp(:   ,Nx  ,2  )=utemp(: ,Nx,1);

error=1;
counter=0;
ts=cputime;
while(error>tol)
counter=counter+1
for i=2:Nx-1
for j=2:Ny-1
utemp(j,i,2)=RHS(j,i)*(1/b)-(a/b)*(utemp(j,i-1,2)+utemp(j,i+1,1))...
-(c/b)*(utemp(j-1,i,2)+utemp(j+1,i,1));
end
end
%SOR
utemp(2:Ny-1,2:Nx-1,2)=   utemp(2:Ny-1,2:Nx-1,2)*omg...
+utemp(2:Ny-1,2:Nx-1,2)*(1-omg);
%Stopping criteria
error=norm(residual(utemp(1:Ny,1:Nx,2),RHS,a,b,c,Nx,Ny))
errora(counter)=error;
if (error>tol)
utemp(2:Ny-1,2:Nx-1,1)=utemp(2:Ny-1,2:Nx-1,2);
else
uout(1:Ny,1:Nx)=utemp(1:Ny,1:Nx,2);
end
end
te=cputime-ts;
end

end
```