MTH 5315 NUMERICAL METHODS FOR PDE
MIDTERM 2

Max Le

ID: 901223283

April 23, 2018

Contents

List of Figures

1 Introduction

For this midterm, we are given the following 1D-Poisson equation to solve:

$$u_{xx} = 1 - 2x^2 \text{ on the domain } (0,1) \tag{1}$$

with Dirichlet boundary conditions as follow: $u(0) = u(1) = 0$. The problem is to be solved using until a relative residual error of 10^{-5} is achieved at a grid size of 128.The results are plotted and the convergent analysis is performed for each case. An analytical solution is also obtained via the following procedure:

$$\begin{aligned}
u_{xx}(x) &= 1 - 2x^2 \\
\frac{d}{dx}(u_x(x)) &= 1 - 2x^2 \\
u_x(x) &= \int_0^x (1 - 2x^2)dx \\
&= x - \frac{2x^3}{3} + C1 \\
u(x) &= \int_0^x (x - \frac{2x^3}{3} + C1)dx \\
&= \frac{x^2}{2} - \frac{x^4}{6} + C1x + C2
\end{aligned}$$

Applying the boundary conditions:

$$u(x) = \frac{x^2}{2} - \frac{x^4}{6} - \frac{x}{3}$$

The 2nd derivative is discretized using a 2nd order central difference scheme as follow:

$$U_{xx} \approx \frac{U_j - 2U_j + U_j}{\Delta x^2} + O(\Delta x^2) \tag{2}$$

This finite difference equation, when combine together with our original PDE, can be written in the form of a linear algebra system, $Au = f$:

$$\frac{1}{\Delta x^2}\begin{bmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \dots \\ f_n \end{bmatrix}$$

With the Dirichlet boundary conditions, our values for f_1 and f_n are 0. There are no need to include ghost points; therefore, our revised linear system is follow:

$$\frac{1}{\Delta x^2}\begin{bmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} 0 \\ f_2 \\ f_3 \\ \dots \\ 0 \end{bmatrix}$$

2 Jacobi Relaxation

2.1 About the method

The general form for an iterative method to solve $Au = f$ is:

$$U^{k+1} = (I - B^{-1}A)U^k + B^{-1}f \tag{3}$$

Where I = identity matrix, D = diagonal vector of A. Also recall that for Jacobi: $B = D^{-1}$, the Jacobi iterative method is as follow:

$$U^{k+1} = (I - D^{-1}A)U^k + D^{-1}f \tag{4}$$

In order to include the under-relaxation feature, the equation needs to be rewitten. The Jacobi method with relaxation parameter, ω, is written as follow:

$$U^{k+1} = \omega[(I - B^{-1}A)U^k + B^{-1}f] + (1 - w)IU^k \tag{5}$$

For our problem, ω is taken to be 0.75.

2.2 Results

Below are results for this method, we have the results converged at 48,906 iterations.

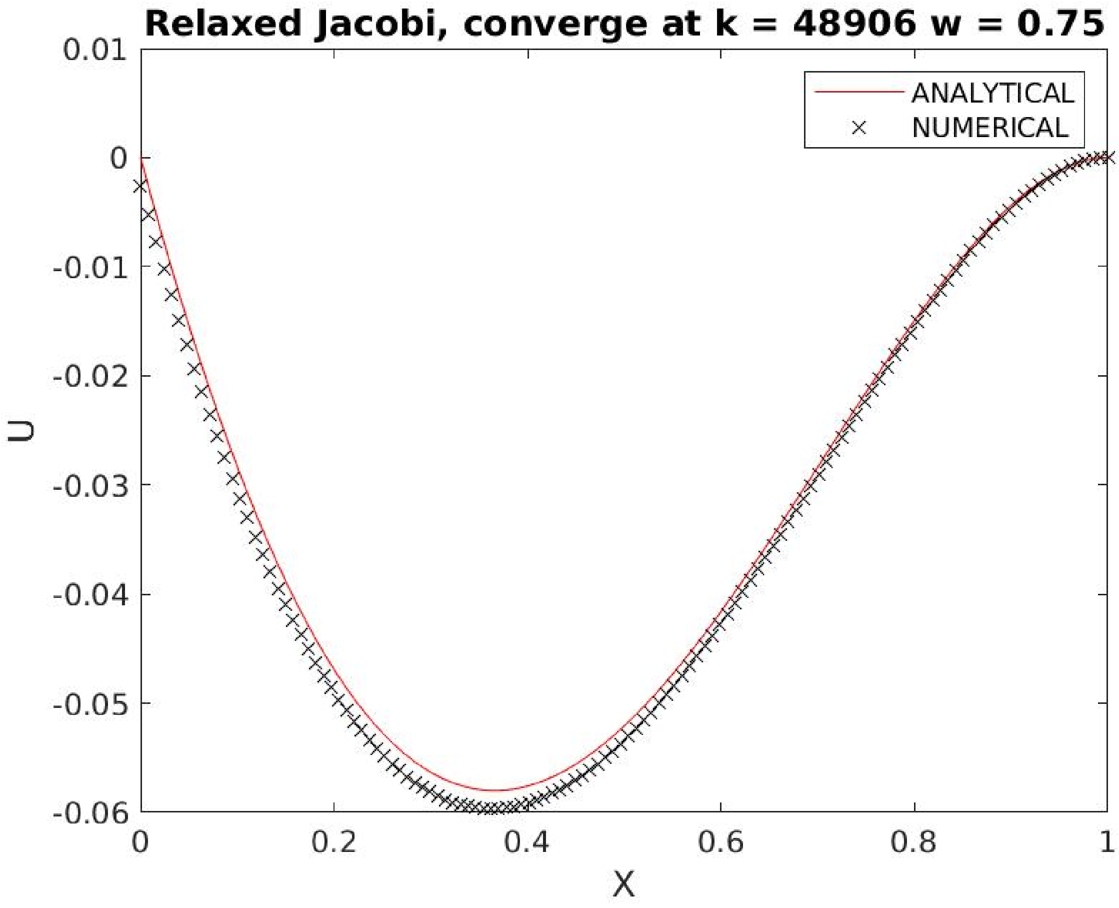

Figure 1: Analytical vs Numerical for Under Relaxed Jacobi for w = 0.75

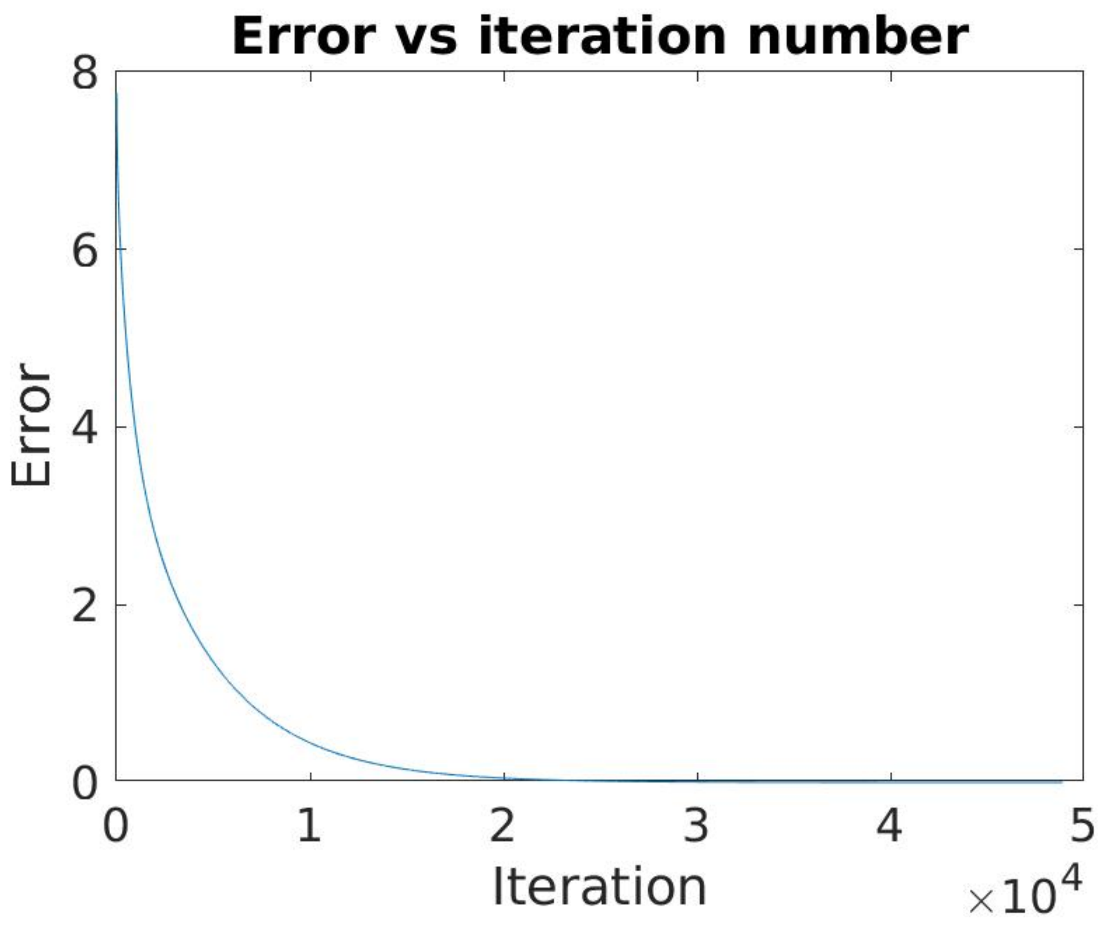

Figure 2: Error vs iteration number for w = 0.75

2.3 Analysis

The Jacobi method is known to be very slow; in this case, we added an under-relaxed parameter, which makes it even slower. The scheme needs almost 60,000 iterations to converge to the analytical solution. Writing in standard form

$$\begin{aligned} U^{k+1} &= Ru^k + c \\ &= \left(\omega(I - D^{-1}A + (1-\omega)I)\right) U^k + wD^{-1}f \end{aligned}$$

The iterative matrix,R, of this method is as follow:

$$R = \omega(I - D^{-1}A) + (1-\omega)I \tag{6}$$

From class notes, for the iterative method to converge, then $||R|| < 1$. Also recall that if R is symmetric then the 2-norm should be used to test for convergence. In this problem, the matrix R looks something like this:

$$\begin{bmatrix} 0.25 & 0.375 & 0 & \dots & 0 \\ 0.375 & 0.25 & 0.375 & \dots & 0 \\ 0 & 0.375 & 0.25 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Its transpose is also the same, therefore we need to use a 2-norm to determine the convergence of this method. This is done by taking the square root of the sum of the squares of all the elements. In the code, the function

norm is used and it gives the answer of **0.998** which is less than 1. Therefore, we can say that this method always converges. Another way to determine convergence is to compute the eigenvalue of R. We can think of the Under Relax Jacobi as a normal Jacobi with some deviations (caused by the relaxation parameter ω). In class, the k-th eigenvalue is computed and it is as follow:

$$\lambda_k = cos\left(\frac{k\pi}{N+1}\right) \text{ , k =1,2,...,N} \tag{7}$$

From trigonometry, cosine is bounded by 1; therefore, $cos\left(\frac{k\pi}{N+1}\right)$ is always bounded 1 and therefore always converges. The convergence rate of the method is dictated by how quickly $||R||$ goes to zero. For an iterative method:

$$\begin{aligned} e^k &= x - x^k \\ &= Re^{k-1} = R(Ae^{k-1}) = ... = R^k e^{(0)} \end{aligned}$$

This can be understood as: the error at after k-th iteration is governed by the value of the matrix R and the initial error. If the 2-norm is taken, then $||e^k|| = ||R^k e^{(0)}|| = ||R^k|| * ||e^{(0)}||$. The convergence rate is therfore controlled by $||R^k||$, which is found out to be **0.998**, which is also ρ- the spectral radius. One thing we can say is that the error rate drops very slow after 1 iteration:

$$\text{Error rate drop} = 1 - 0.998 = 0.002 = 0.2\%$$

Therefore, the Under Relaxed Jacobi, when applies to this problem, gives converged solution. However, the rate of convergence is very slow. We need at least 60,000 iterations to satisfy a tolerance of 10^{-5}. As an additional investigation, if the value of ω is changed, then we have:

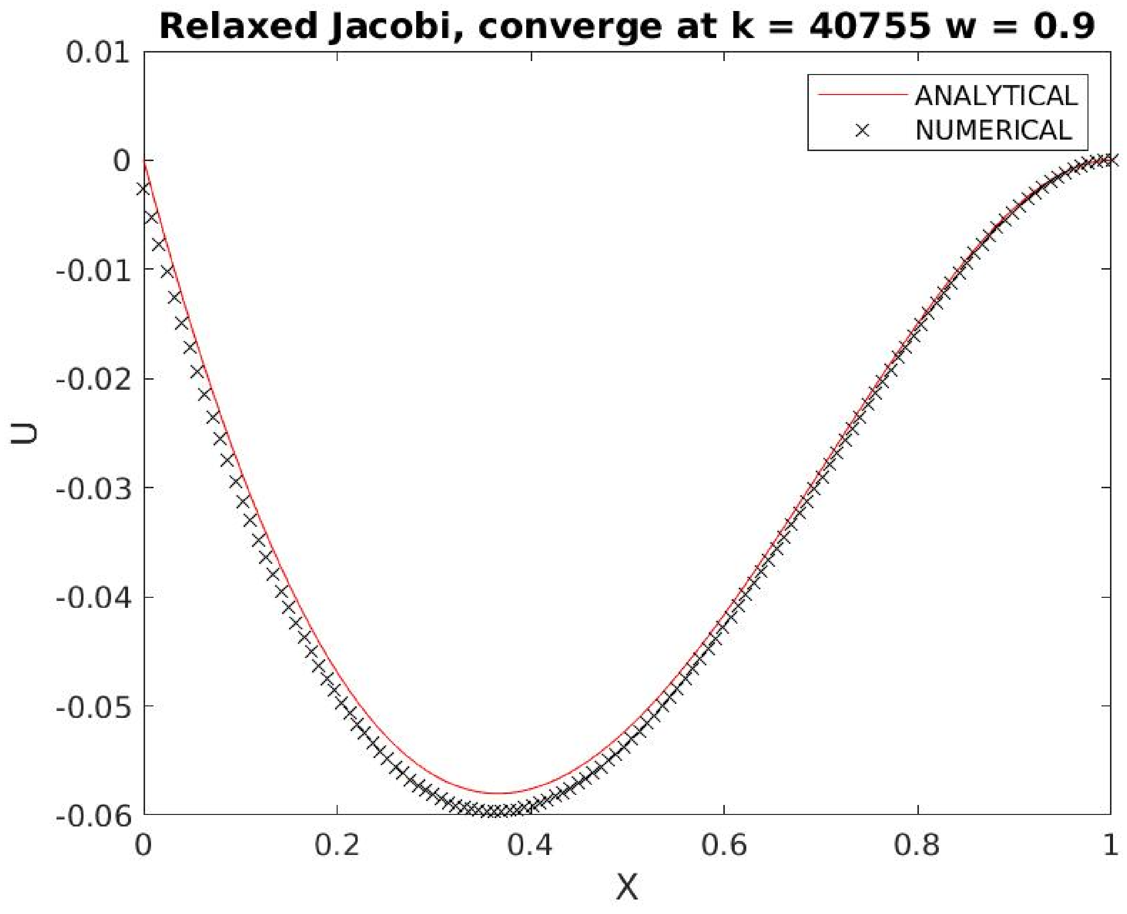

Figure 3: Analytical vs Numerical for Under Relaxed Jacobi for w = 0.9

Clearly,the number of iterations decrease as ω goes up. However, if we keep increasing:

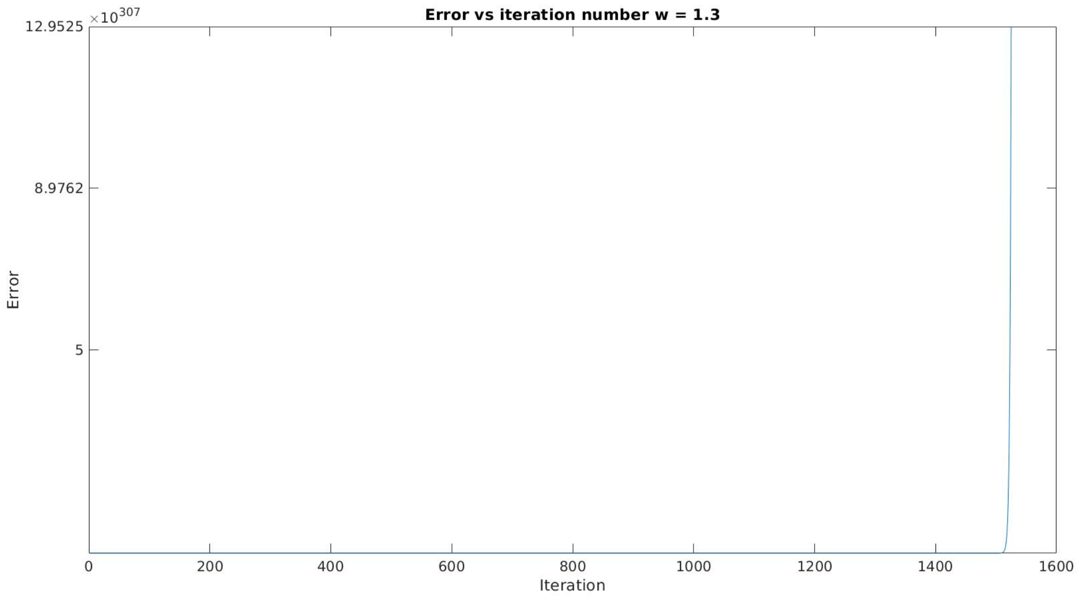

Figure 4: Error vs iteration number for w = 1.3

The error blows up too much that the scheme is unable to find a converge solution. In our error equation, the iterative error term e^k is too large for the iterative matrix R^k and the initial error $e^{(0)}$ to contain and reduce.

3 Multigrid V-Cycle

3.1 About the method

The basic idea of the Multigrid method is to remove errors at low frequencies. Gauss-Seidel and Jacobi method and their error vector,e, has most of the high frequency errors removed but not the low frequency ones. These errors are usually not removed because they are in the ”smooth” region. In order to do this, we change from a fine grid to a coarser grid, such that smooth errors become rough errors and as a result, the low frequency errors will appear as high frequency errors and hopefully, will be removed by the usage of Gauss-Seidel/Jacobi.. After the errors are removed, we then convert the coarser grid back into the fine grid and get our results. This ”return step” is done by an interpolation matrix I^h_{2h}.

Our problem requires a 4 level of grid coarsening V-cycle multigrid. As a result, the top ”fine” grid will have 128 elements, the next coarse grid will have 64, then 32, then 16 and 8 elements. A visual for this V-cycle method can be shown below:

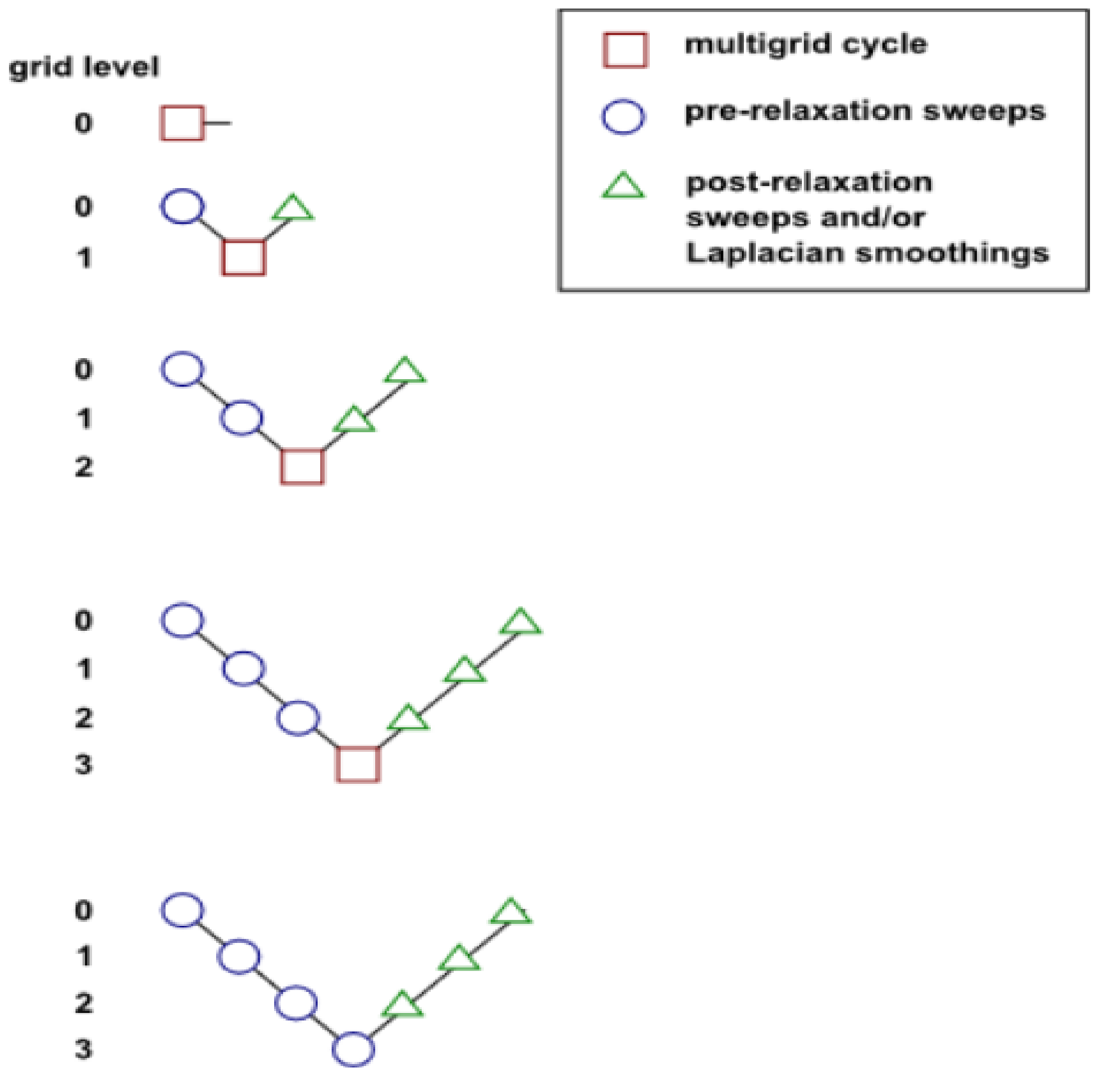

Figure 5: Error vs iteration number

3.2 Results

Below are the results obtained for this method. We have convergent at 21 iterations.

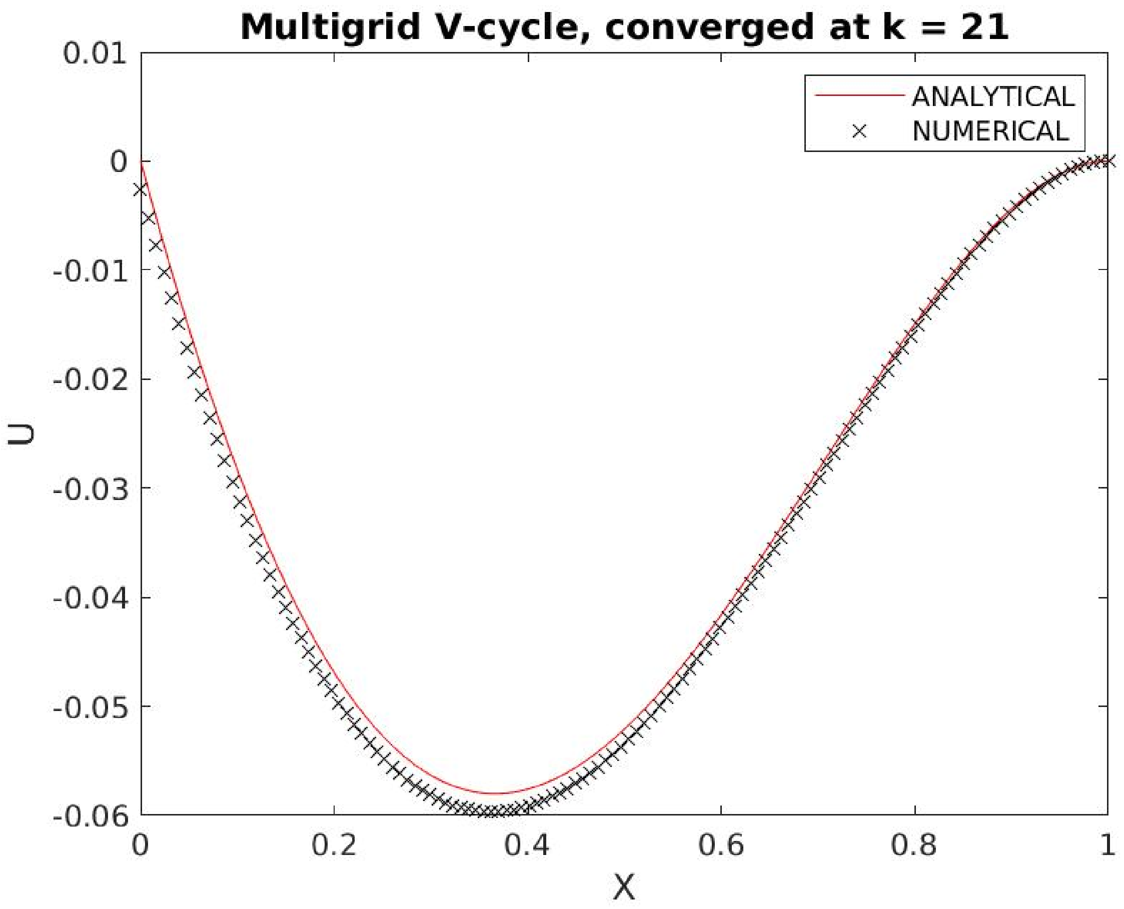

Figure 6: Analytical vs Numerical for 4-Level V-cycle Method

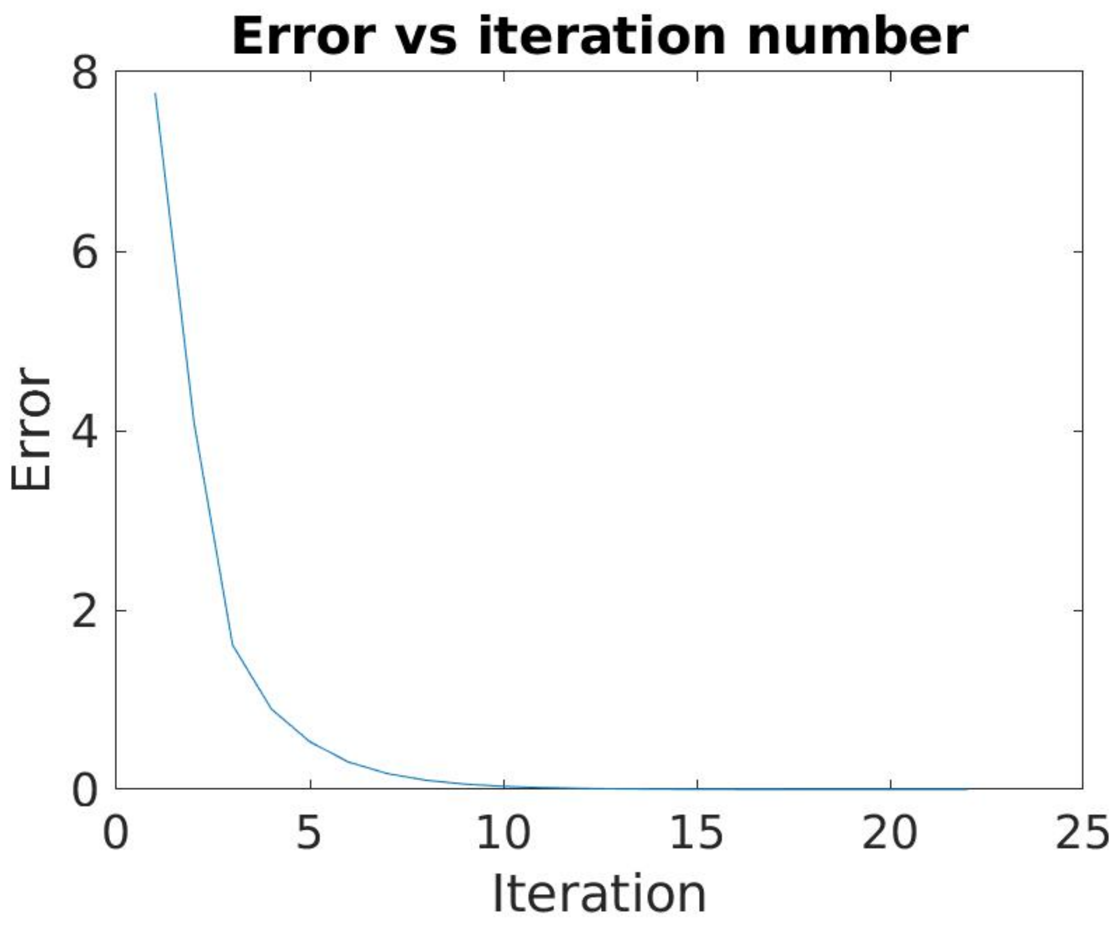

Figure 7: Error vs Iteration number for 4-Level V-cycle Method

3.3 Analysis

It can be seen that the Multigrid approach is extremely efficient. From before, we need almost 60,000 iterations to converge. Now, we only need around 10 iterations to converge. A similar analysis containing the kth level error and initial error can also be done:

$$e^k \leq \rho e^0 \tag{8}$$

Where ρ is the constant factor to govern the convergence rate of the method that is independent of the grid size. In the code, this ρ is calculated to be: $1.22 * 10^{-6}$. This is significantly smaller than the value we calculated for Jacobi, which was 0.999. Using the same analysis, after 1 iteration, the error gets reduced by: $1 - 1.22 * 10^{-6} = 0.99999878 = 99.9\%$. These are all estimates because rigorous analysis of the convergence rate are not discussed in class. However, it still give us an idea of how fast and efficient the Multigrid V-cycle method is.

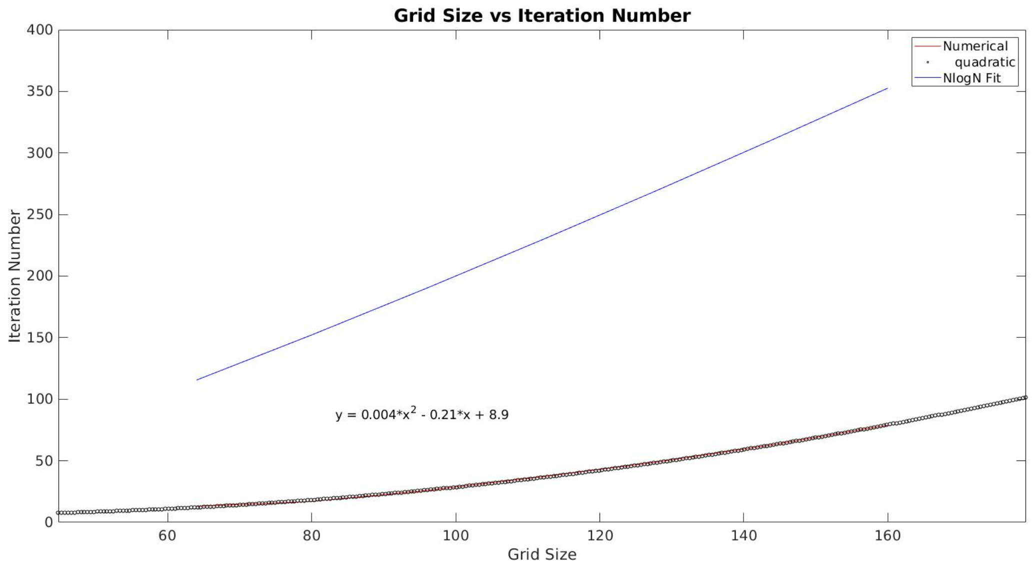

Figure 8: Grid Size vs Iteration number for 4-Level V-cycle Method

The plot above shows the grid size vs the number of iteration it needs to converge for our particular problem. It can be seen as grid size increases, we need more iterations. This is understandable because the larger problem contains more unknowns and therefore more equations are needed to be solved. Theoretically, the curve $NlogN$ can be used to fit our numerical data. However, this is only estimate because we have not really covered the diffrent parameters such as y-intercept,x-intercept of the $NlogN$ curve for this method. Instead, another polynomial curve is fitted to show the general trend. This is a 2nd order polynomial and if we think about it, the result makes sense. The reason for this is that our original PDE is discretized using a 2nd order central difference with a local truncation error of Δx^2. Therefore, if we "square" our gridsize, then the error should also increase which makes the number of iterations needed to also increase. They will following a 2nd order polynomial curve (x^2)

4 Conclusion

In conclusion, the Under Relaxed Jacobi method shows how the value of the relaxation parameter, ω, is important in controlling the convergence rate. Good choice of ω will speed up the convergence rate;however, poor choice of ω will cause the numerical solution to blow up. On the other hand, the Multigrid V-cycle involves a recursive algorithm to take care of the errors at low frequency. This is done by mapping the problem from an original (fine) grid to a coarser grid, with the hope that the errors at low frequency becomes high frequency and therefore can be reduced by the usage of Jacobi/Gauss-Seidel..etc. The result was significantly better than Under Relaxed Jacobi, we managed to get a much lower iteration numbers to satisfy the same tolerance. From the error plots, we can also see clearly how the errors for the V-cycle drop to around 0 after 5 iterations, compare to Under Relax Jacobi, where the we need at least 20,000 iterations to get a "low" error. Finally, the comparison between Grid Size and Iteration Number shows how the scheme follows a quadratic increase in terms of iteration number and error. If more info is given, then the curve $NlogN$ can be fitted better to show that the scheme follows an $NlogN$ order.

5 Matlab Codes

5.1 Under Relaxed Jacobi

```
clear all
clc
% clf

%FORM:  uxx = -f = g

%DEFINE GRID SIZE
a =0; %first grid point
b = 1; %last grid point
jmax = 128; %max number of cells
dx = (b-a)/(jmax-1); %spacing


%DEFINE BASIC PARAMETERS
tol = 1e-5; %error
x = linspace(a,b,jmax); %mesh space
unew = zeros(1,jmax); %solution
%right hand side vector
g = 1-2.*(x'.^2);

%initial conditions
u = 0*(x');
u(1) = 0;
u(jmax) = 0;

uold = u;

%ANALYTICAL SOLN

u_analytical = ((x.^2)/2)-((x.^4)/6)-(1/3)*x;

A = full(gallery('tridiag',jmax,1,-2,1));

A = (1./dx^2)*A;

%
rhs = (dx^2).*(g);
%
ureal = (A\(rhs));
ureal = ureal.';

%getting diagonal vectors

D = eye(jmax)*(-2/dx.^2);
Lt = tril(A);
L = Lt - D;
%

omega = 0.90;

I = eye(jmax);

u_jacobi = zeros(jmax,1);
u_updated = u_jacobi;
Dinv = inv(D);

res = g - (A*u_jacobi);

err_v = norm(res);

err_start  = norm(res);

```

```

for k = 1:100000
    u_updated = omega*((I-Dinv*A)*u_jacobi+Dinv*g)+(1-omega)*u_jacobi;
    u_jacobi = u_updated;
    res = g-(A*u_updated);
    err_v = [err_v,norm(res)];
    if (norm(res)/norm(g) < tol)
        break;
    end

end


%% CONVERGENCE ANALYSIS

% Binv = Dinv;
%
% R_jacobi = eye(jmax)-Binv*A;
% max(abs(eig(R_jacobi)))
% R_iterative = omega*(eye(jmax)-Binv*A)+(1-omega)*eye(jmax);
% disp(size(R_iterative));
% Rtrans = R_iterative.';
% norm_R_iter = norm(R_iterative);
% disp(norm_R_iter)
%
% %spectral radius
%
% spec_rad = max(abs(eig(R_iterative)));
% disp(spec_rad);


%% PLOTTING

% f1 = figure(1);
% plot(x,u_analytical,'-r')
% hold on
% plot(x,u_jacobi,'xk')
% title(['Relaxed Jacobi, converge at k = ' num2str(k) ' w = ' num2str(omega)],'FontSize',12)
% xlabel('X','FontSize',12)
% ylabel('U','FontSize',12)
% xt = get(gca, 'XTick');
% legend('ANALYTICAL','NUMERICAL')
% saveas(f1,'jacobi75.jpg')
%
% %
% f2 = figure(2);
% plot((err_v));
% title('Error vs iteration number','FontSize',12);
% xlabel('Iteration','FontSize',24)
% ylabel('Error','FontSize',24)
% xt = get(gca, 'XTick');
% set(gca, 'FontSize', 16)
% saveas(f2,'jacobi_error.jpg')
```

5.2 Multigrid V-Cycle

```

clear all
clc

jmax = 128;


x = linspace(0,1,jmax);
x = x.';
u = zeros(1,jmax);
u = u.';
g = 1-2.*(x'.^2);
g = g.';

dx = (1-0)/(jmax-1);

A = full(gallery('tridiag',jmax,1,-2,1));


A = (1./dx^2)*A;

u = zeros(jmax,1);
res = g-(A*u);

err_v = norm(res);

tol = 1e-5;

kmultigrid = 5000;

err_start = norm(res);
for k = 1:kmultigrid
    u = vfunct(u,g,8*dx,1,0);
    res = g-(A*u);
%     disp(norm(res));
    err_v = [err_v, norm(res)];
    if (norm(res)/norm(g)<tol)
      break;

    end
end

disp(k);

err_end = norm(res);

rho = err_end/err_start;

disp(rho);


u_analytical = ((x.^2)/2)-((x.^4)/6)-(1/3)*x;

f1 = figure(1);
plot(x,u_analytical,'-r')
hold on
plot(x,u,'xk')
title(['Multigrid V-cycle, converged at k = ' num2str(k)],'FontSize',12)
xlabel('X','FontSize',12)
ylabel('U','FontSize',12)
xt = get(gca, 'XTick');
legend('ANALYTICAL','NUMERICAL')
saveas(f1,'vcycle.jpg')

```

```
%
% f2 = figure(2);
% plot((err_v));
% title('Error vs iteration number','FontSize',12);
% xlabel('Iteration','FontSize',24)
% ylabel('Error','FontSize',24)
% xt = get(gca, 'XTick');
% set(gca, 'FontSize', 16)
% saveas(f2,'vcycle_error.jpg')




%% FUNCTION FOR V-CYCLE

function vcycle = vfunct(u,f,maxdx,xmax,xmin)


s = size(u);
jmax = s(1);
dx = (xmax-xmin)/(jmax-1);
x = linspace(xmin,xmax,jmax);


%% DEFINE BASIC PARAMETERS
tol = 1e-5; %error

A = full(gallery('tridiag',jmax,1,-2,1));


A = (1./dx^2)*A;



%% RELAX, DO JACOBI


D = eye(jmax)*(-2/dx.^2);


omega = 0.75;
B =D;
Binv = inv(B);


%
R = eye(jmax)-Binv*A;


u_relax = u;
c = Binv*(f);
res = f - (A*u_relax);
err_v = norm(res);


for k = 1:20

  u_new = (R*u_relax)+c;
  u_relax = omega*u_new+(1.-omega)*u_relax;

  res = f-(A*u_relax);
end

%% CHECKING

if (dx ≥ maxdx)
  vcycle = u_relax;
```

```
else
  r2h = zeros(jmax/2,1);
  for i = 1:jmax/2
    r2h(i) = res(2*i);
  end
  u2h = zeros(jmax/2,1);

  u2h = vfunct(u2h,r2h,maxdx,1,0);

  x2 = linspace(xmin,xmax,jmax/2);

  %INTERPOLATION
  interp_result = interp1(x2.',u2h,x.');

  u_relax = u_relax + interp_result;
%   disp(size(u_relax));

  %% RELAX AGAIN

  res = f - (A*u_relax);
  err_v = norm(res);

  for k = 1:20

    u_new = (R*u_relax)+c;
    u_relax = omega*u_new+(1.-omega)*u_relax;
    res = f-(A*u_relax);

   end

end


vcycle = u_relax;



end
```