Numerical PDE II HW 2 : Elliptic Equations on the Square Part 2: Multigrid

Dhruv Balwada

February 26, 2015

1 Statement of Problem

In this problem we will explore the Multigrid method for the solution of the Dirichlet problem for the PDE

$$u_{xx} + u_{yy} = f; x \in \omega \tag{1}$$

$$u(x) = g(x); x \in \partial\omega \tag{2}$$

for $\omega = [0, 1] \times [0, 1]$. The laplacian is approximated by the standard second order five point stencil approximation.

2 Description of the Mathematics

We are required to solve an elliptic equation on the square. Here we will approximate the laplacian operator using a second order difference approximation

$$\nabla_h U = (\delta_x^+ \delta_x^- + \delta_y^+ \delta_y^-)U = F_{ij} \tag{3}$$

where the δ operators are the standard notation. This is a 5 point stencil. We show that this approximation is second order on attached hand written notes.

To solve this system on a square grid we need to use an iterative solver. In this exercise we will use the Multigrid method and then compare it to the preconditioned conjugate gradient method.

How and why does the algorithm work? The multigrid algorithm takes advantage of the fact that the simple iterative solvers(Jacobi, Gauss-Seidel etc.) show extremely fast reduction of errors at the high frequencies but are slower when it comes to low frequency errors. Also it is noted that high frequency errors on a coarse mesh are low frequency on a fine mesh. Using this idea the Multi-grid algorithm essentially relaxes for a few iterations using an iterative solver on the fine grid by which point the high frequency errors have been damped. Then it restricts the current solution state to a coarser mesh and solves for a few more iterations. This process is repeated again and again always stepping to coarser meshes.

At the coarsest mesh the solution is stepped back to the finest mesh, again relaxing in the process. It should be noted that the algorithm that steps all the way from the finest to coarsest and then all the way back up is the V-cycle algorithm. Other algorithms are also possible that don't necessarily take all the steps down and back in a continuous fashion. However we don't discuss them here as we will be working with the V-cycles algorithm.

3 Description of the Algorithm

Here I present the algorithms and describe if necessary the main code segments that contribute to the method.

3.1 Multigrid main

The main function that calls other functions and carries out the skeleton of the V-cycles.

Initialize GRID(M), where M is the number of grids
Assign boundary conditions, forcing on finest grid, X, Y etc.
Initialize residual =1.0, $tol = 10^{-}10$, ld, lu
while $tol < residual$ **do**
  for k=M to 1 **do**
    Relax $L_k U^k = F^k$ l_d^k times
    if $(k > 1)$ **then**
      CALL FtoC
    end if
  end for
  for k=2 to M **do**
    CALL CtoF
    Relax $L_k U^k = F^k$ l_u^k times
  end for
end while

3.2 FtoC

The code to step solution from a finer to coarser mesh.

Initialize the required variables and receive data from main program.
do $U^{k-1} = R_k^{k-1} U^k$ by calling the restriction sub routine.
Calculate $L_{k-1} U^{k-1}$
Calculate $F^k - L_k U^k$
Call restriction sub routine on $F^k - L_k U^k$ that was just calculated.
Compute $F^{k-1} = L_{k-1} U^{k-1} + R_k^{k-1}(F^k - L_k U^k)$

3.3 CtoF

The code to step solution from a coarser to a finer mesh.

Initialize the required variables and receive data from main program.
Compute $U^{k-1} - R_k^{k-1}U^k$ by calling restriction and subtraction.
Interpolate the quantity(error) just calculated to a finer grid by using interpolation subroutine.
Computer $U^k = U^k + error$.

3.4 relax gauss seidel

This is the relaxation routine

Initial local variables. Get the array that needs to be relaxed. resmax=0.0
for all interior grid points call resfor sub routine to calculate residuals.
$U(i,j) = U(i,j) - 1/4 * h^2 * residual$
Also find resmax = max(abs(residual),resmax)

3.5 resfor

This is the subroutine to calculate the residuals for the Gauss-seidel relaxer.

Obtain forcing and current solutions arrays
calculate resout = forcing - L(U)

3.6 interpolation

This is the interpolation routine that uses bilinear interpolation

Initial required variables and obtain the coarse array.
Copy the corner points as is Uf(2*j,2*i) = Uc(j,i)
For half way points in columns Uf(j,i) = 0.5d0*(Uf(j-1,i)+Uf(j+1,i))
For half way points in rows Uf(j,i) = 0.5d0*(Uf(j,i-1)+Uf(j,i+1))
For middle of grid points Uf(j,i) = 0.25d0*(Uf(j,i-1)+Uf(j,i+1)+Uf(j-1,i)+Uf(j+1,i))

3.7 restriction

This is the restriction routine that uses point wise restriction.

Simply do Uc(j,i) = Uf(2*j,2*i)

4 Results

Similar to the assignment 1, I will solve the following system:

$$\nabla u = -2cosxsiny \tag{4}$$

The boundary conditions on the square are $cosxsiny$.

In figure 1 the behavior of max norm of error with changing h(mesh size) is shown. The order of the scheme is 2 as we expect. As the scheme is working as expected we go ahead and analyze the behavior of the multigrid solver.

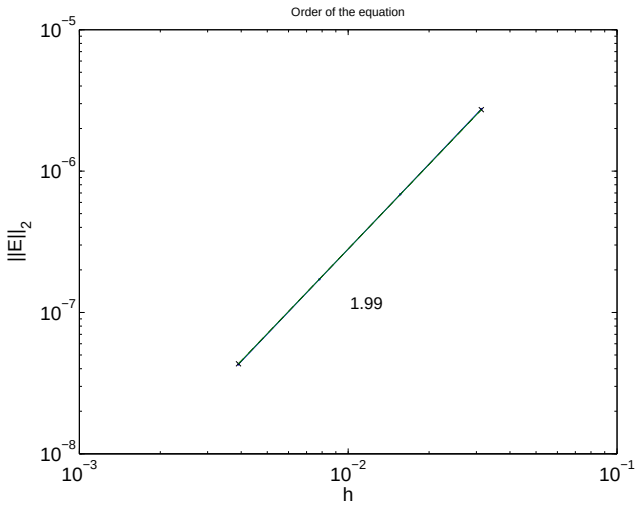

Figure 1

In figure 2 the max norm of the residual vs the number of V-cycles is plotted in a semi-log plot. It is clear that the residual nrom decreases exponentially with V-cycles. We calculate the slope of these lines to be -0.75 or -3/4.

$$||R|| = ||R_o||10^{-3/4n}$$

Where R is the residual, R_o is the initial residual and n is the number of v-cycles. We can use this simple formula to predict the number of V-cycles it will take. The formulation might be applicable only for this test case but probably similar formulations exist for all possible cases. The interesting point is that one can calculate the decay the scale from the coarsest run and then use it to predict the number of V-cycles to get to a required tolerance for a larger mesh if we know the initial residual.

$$n = -4/3log_{10}(||tolerance||/||R_o||)$$

Is the number of V-cycles required to go to a specified tolerance. For example for the case of 256 size mesh we know that the $||R_o||$ 10^5. To get to a tolerance of 10^-10 we will need

$$n = -4/3log_{10}(10^{-10}/10^5) = 20$$

Similarly to get a tolerance of 10^{-5} we will need

$$n = -4/3log_{10}(10^{-5}/10^5) = 40/3 = 13.3$$

We see that these predictions from the formula match up well with the results in the graph. So now we have a relation for the residual behavior for different tolerance values.

The above analysis was done for ld and lu, which are the number of relaxation steps in every step of the V-cycle, of 1. In Table 1. the results for different values of ld and lu are presented. The main results are that the number of V-cycles required decrease with increasing number of relaxation steps, however the time taken is not significantly affected. The minimal changes in run time show that choosing ld and lu of 2 is the fastest.

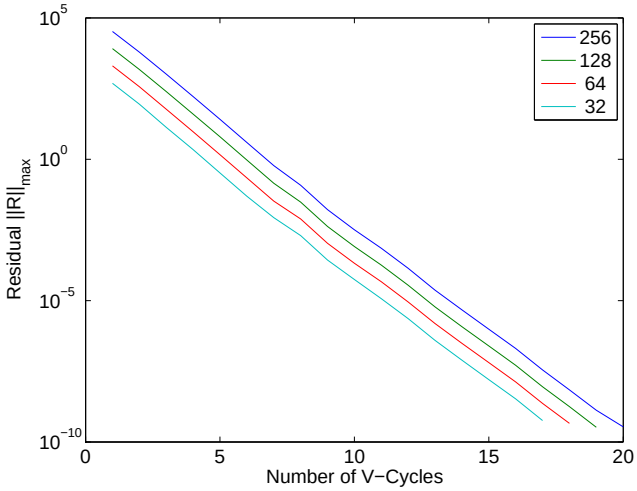

Figure 2

In the same table the run times from the Preconditioned conjugate gradient solver are also presented. The multigrid solver is faster by about 5 times for the 256 mesh and about 3 times for the 128 mesh. This pattern suggests that the multigrid solver is an excellent choice for larger mesh sizes.

	Multigrid(ld,lu =1) Time	Multigrid(ld,lu =1) V-cycles	Multigrid(ld,lu =2) Time	Multigrid(ld,lu =2) V-cycles	Multigrid(ld,lu =3) Time	Multigrid(ld,lu =3) V-cycles	Preconditioned
256	0.340143	20	0.26342	10	0.293266	8	1.29
128	0.0791	19	0.065558	10	0.072363	8	0.17
64	0.019022	18	0.014775	9	0.017878	8	.0218
32	0.004787	17	0.003846	9	0.00399	7	0.00486

5 Conclusions

The Multigrid algorithm was implemented, tested and analyzed. Even though the implementation complexity is increased the gains in run times are impressive. The rate of convergence to the solution is exponential with a rate that is independent of the grid size and formula is presented.