Assignment 02

ME 670: Advanced Computational Fluid Dynamics

Multi-Grid Methods

V-Cycle and Full-Multigrid Method

Nirmal S.
[234103107]
Computational Mechanics
MTech. Mechanical Engineering
IIT-Guwahati

Table of Contents

Problem Statement:

1. Develop a V-cycle program for the one-dimensional model problem $-u"(x)+\sigma u(x)=f$, with homogenous boundary conditions, solved using finite difference method. Write a function/subroutine for each individual component of the algorithm as follows.

(a) Given an approximation array v, a right-side array f, and a level number $1\leq l\leq L$ (smallest level number corresponds to the finest grid), write separate subroutines that will carry out ν number of weighted Jacobi or Gauss-Seidel sweeps on level l. Keep them in a file named "relaxation_methods" for example: relaxation_methods.F90
(b) Given an array f and a level number $1\leq l\leq L-1$, write a subroutine that will carry out full weighting between level l and level $l+1$. Keep it in a file named "restriction_methods".
(c) Given an array v and a level number $2\leq l\leq L$, write a subroutine that will carry out linear interpolation between level l and level $l-1$. Keep it in a file named "prolongation_methods".
(d) Write a subroutine named 'V_cylce' that carries out a single V-cycle by calling the three preceding subroutines. The V-cycle should be able to start from a given level l. Keep it in a file named "MG_methods".
(e) Write a main program that initializes the data arrays and calls V-cycle subroutine. For testing, for fixed k, take $f(x)=C\sin(k\pi x)$ on the interval $0\leq x\leq 1$, where C is a constant. Then the exact solution to model problem is

$$u(x) = \frac{C}{\pi^2k^2+\sigma}\sin(k\pi x)$$

(f) Write another subroutine that computes 2-norm of error and residual. Keep it in a file named "postprocessing_methods". You can also keep files writing subroutines in this file.
(g) Take $n=512$, $\omega=2/3$, $\nu1=\nu2=2$, $\sigma=1$ and $C=\pi 2k2+\sigma$. For $k=1$ and 10, apply basic iterative methods and V-cycle iterations till the residual 2-norm is greater than $10-6$. Plot the residual norm against the number of iterations for all three methods on the same figure.

2. Using the V-cycle subroutine, write a full multigrid (FMG) subroutine named 'FMG'. The FMG-cycle should start from a given level l. Keep it in the file named "MG_methods". Verify the solver using the problem statement given in Q1. Report the 2-norm of the residual after applying one FMG-cycle to the test problem in Q1. Afterwards, apply V-cycle to solution approximation till the residual 2-norm is greater than 10^{-6}. Report the number of V-cycle iterations.

Grid Details

A 1D grid is considered, which stores the values of *u, v, f, v_temp* which can be split in levels as shown in the figure below:

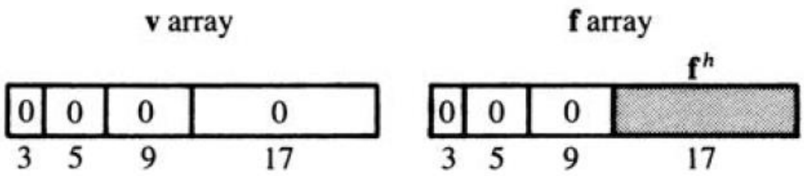

Figure 1: Image of the arrays to store the values (here the finest grid is considered with 17 grid points).

If the number of grid points are 2^l+1: (for 512 divisions, there are 513 grid points and l=9)

Then the total length of the array: $(2^1+1)+(2^2+1)+\cdots+(2^l+1)=(2^{l+1}-2)+l$

Number of grids in level l: 2^l+1

For a given number of divisions, number of levels *Levels*: $\log_2(no\_of\_div\ +1)$

Discretised Equations

The 1D model problem:

$$-u''(x) + \sigma u(x) = f$$

Discretising it in Finite Difference Method:

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + \sigma u_i = f_i$$

C-Code: Multi Grid Methods

Main Code:

The main function will run either V-Cycle or Full-MultiGrid based on the user input. The function can be checked from the program file *"A02_234103107.c"*.

Functions:

Functions to calculate the starting point of the level, and the number of cells in the array at that level:

```
int index_level(int Levels, int lvl){return pow(2, Levels-lvl+1) - 2 + (Levels-lvl);}

int num_grid(int Levels, int lvl){return pow(2,Levels-lvl+1) + 1;}
```

Function to perform relaxation method, based on the level *lvl* at which it has to be performed, on array *v* and *f*, and based on the *method* which is *J* for Jacobi and *G* for Gauss Seidel.

```
void relaxation_methods(int M, int lvl, int Levels, double v[M], double f[M], double sigma, int nu,
                        char method, double J_weight, double h){

        int index_lvl = index_level(Levels, lvl);
        int m = index_lvl + num_grid(Levels, lvl);
        double h2 = pow(h,2);

        if(method == 'J') {
                int n = pow(2,Levels-lvl+1) + 1;
                double v_new[n];
                for(int i=0; i<n; i++) v_new[i] = v[index_lvl+i];

                for(int iter=0; iter<nu; iter++){
                        for(int i=1; i<n-1; i++){
                                v_new[i] = (1-J_weight)*v_new[i] + J_weight*(h2*f[index_lvl+i] + 
v[index_lvl+i-1] + v[index_lvl+i+1])/(2+sigma*h2);
                        }
                        for(int i=0; i<n; i++)  v[index_lvl+i] = v_new[i];
                }
        }

        if(method == 'G') {
                for(int iter=0; iter<nu; iter++){
                        for(int i=index_lvl+1; i<m-1; i++){
                                v[i] = (h2*f[i] + v[i-1] + v[i+1])/(2+sigma*h2);
                        }
                }
        }

        return;
}
```

Function to perform *Full Weighting* restriction method to *f* at level *lvl* and store it in the next level.

```
void restriction_methods(int M, int lvl, int Levels, double f[M]){

        int index_lvl = index_level(Levels, lvl);
        int index_lvl_nxt = index_level(Levels, lvl+1);
        int n = num_grid(Levels, lvl);

        for(int i=1; i<n/2; i++){
                f[index_lvl_nxt+i] = 0.25*(f[index_lvl+2*i-1] + 2*f[index_lvl+2*i] +
                                           f[index_lvl+2*i+1]);
        }
        return;
}
```

Function to perform *Linear Interpolation* prolongation method to *v* at level *lvl* and store it in the previous level

```
void prolongation_methods(int M, int lvl, int Levels, double v[M], double v_temp[M]){

      void v_Boundary_Conditions(int m, int n, double v[m][n+1], double v_left_value, double v_right_value,
                                 double v_top_value, double v_bottom_value){
         for(int i=0; i<m; i++){
                 v[i][0] = v_left_value*2 - v[i][1];
                 v[i][n] = v_right_value*2 - v[i][n-1];
         }
         for(int j=0; j<n+1; j++){
                 v[0][j] = v_top_value;
                 v[m-1][j] = v_bottom_value;
         }
         return;
      }
}
```

Function to calculate error and residual at level *lvl* using *u, v,*and *f* and store it in *error* and *res*.

```
void post_processing_methods(int M, int lvl, int Levels, double h, double sigma, double f[M], double
v[M], double u[M], double *error, double *res){

        int index_lvl = index_level(Levels, lvl);
        int n = num_grid(Levels, lvl);
        double h2 = pow(h,2);

        for(int i=index_lvl+1; i<index_lvl+n-1; i++){
                *res += pow(f[i] - 1/h2*(- v[i-1] - v[i+1] + (2+sigma*h2)*v[i]), 2);
                *error += pow(u[i] - v[i], 2);
        }
        *res = sqrt(*res);
        *error = sqrt(*error);

        return;
}
```

Function to run *V-Cycle*

```
void v_cycle(int M, int Levels, int lvl, double v[M], double f[M], double residual[M], double u[M],
              double C, double k, double sigma, int nu1, int nu2, double h, char method, double J_weight){

         double v_temp[M]; // temp variable used in V-cycle while going up the levels
         for(int i=0; i<M; i++) v_temp[i] = 0;
         for(int i=0; i<index_level(Levels, 1); i++) v[i] = 0;

         int index_lvl = index_level(Levels, lvl);
         int n = num_grid(Levels, lvl);
         double h1, h2;

         // *** Initialising 'f' (RHS) array and exact solution 'u' ***
         for(int i=index_lvl; i<index_lvl+n; i++) f[i] = C*sin((float)k*M_PI*(i-index_lvl)*h);
         for(int i=index_lvl; i<index_lvl+n; i++) u[i] = C/(pow(M_PI,2)*pow(k,2) +
                                                   sigma)*sin((float)k*M_PI*(i-index_lvl)*h);


         // *** Moving towards coarser grids ***
         for(int lvl_i=lvl; lvl_i<Levels; lvl_i++){

                  h1 = h*pow(2,lvl_i-1);
                  h2 = pow(h1,2);

                  // Applying relaxation method to next level
                  relaxation_methods(M, lvl_i, Levels, v, f, sigma, nu1, method, J_weight, h1);

                  // Calculating the residual
                  index_lvl = index_level(Levels, lvl_i);
                  n = num_grid(Levels, lvl_i);
                  for(int i=index_lvl+1; i<index_lvl+n-1; i++)    residual[i] = f[i] - 1/h2*(- v[i-1] -
                                                                      v[i+1] + (2+sigma*h2)*v[i]);

                  // Taking residual to next level
                  restriction_methods(M, lvl_i, Levels, residual);
                  index_lvl = index_level(Levels, lvl_i+1);
                  n = num_grid(Levels, lvl_i+1);

                  // Storing residual back into 'f' at the next level
                  for(int i=index_lvl+1; i<index_lvl+n-1; i++)   f[i] = residual[i];

         }


         // *** Solve the equation at the coarsest grid ***
         h1 = h*pow(2,Levels-1);
         relaxation_methods(M, Levels, Levels, v, f, sigma, nu1, method, J_weight, h1);


         // *** Moving towards finer grids ***
         for(int lvl_i=Levels; lvl_i>lvl; lvl_i--){

                  // Interpolate to the previous level
                  prolongation_methods(M, lvl_i, Levels, v, v_temp);

                  // Recalculate the 'v' value with the interpolated value
                  index_lvl = index_level(Levels, lvl_i-1);
                  n = num_grid(Levels, lvl_i-1);
                  for(int i=index_lvl+1; i<index_lvl+n-1; i++)   v[i] += v_temp[i];

                  // Relaxing the equation at the previous level
                  h1 = h*pow(2,lvl_i-2);
                  relaxation_methods(M, lvl_i-1, Levels, v, f, sigma, nu2, method, J_weight, h1);

         }

         return;
}
```

Results

1. Plotting residual norm against number of iterations:

For $k = 1$:

For $k = 10$:

2. Report of residual after FMG and number of V-cycles needed for until residual norm is below epsilon

K	Relaxation Method	Residual after one FMG cycle.	Number of V-Cycles required further
1	Gauss Seidel	18.8597	6
1	Jacobi	25.3194	8
10	Gauss Seidel	1843.9372	7
10	Jacobi	2133.6977	9