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: 2](#_Toc162821389)

[Grid Details 3](#_Toc162821390)

[Discretised Equations 4](#_Toc162821391)

[C-Code: Multi Grid Methods 5](#_Toc162821392)

[Results 8](#_Toc162821393)

# Problem Statement:

**1**. Develop a V-cycle program for the one-dimensional model problem −𝑢"(𝑥)+𝜎𝑢(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≤𝑙≤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

1. (b) Given an array 𝑓 and a level number 1≤𝑙≤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”.
2. (c) Given an array 𝑣 and a level number 2≤𝑙≤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”.
3. (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”.
4. (e) Write a main program that initializes the data arrays and calls V-cycle subroutine. For testing, for fixed 𝑘, take 𝑓(𝑥)=𝐶sin(𝑘𝜋𝑥) on the interval 0≤𝑥≤1, where *C* is a constant. Then the exact solution to model problem is
5. (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.
6. (g) Take 𝑛=512, 𝜔=2/3, 𝜈1=𝜈2=2, 𝜎=1 and 𝐶=𝜋2𝑘2+𝜎. 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 . 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:

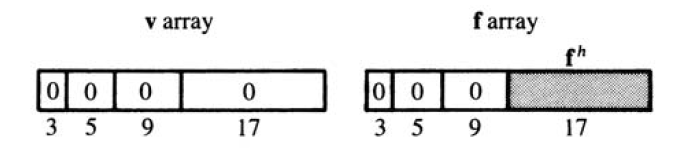


Figure 1: Image of the arrays to store the values (here the finest grid is considered with 17 grid points).

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

Then the total length of the array:

Number of grids in level *l*:

For a given number of divisions, number of levels *Levels*:

# Discretised Equations

The 1D model problem:

Discretising it in Finite Difference Method:

# 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]){

int index\_lvl = index\_level(Levels, lvl);

int index\_lvl\_prev = index\_level(Levels, lvl-1);

int n = num\_grid(Levels, lvl-1);

for(int i=0; i<n/2; i++){

v\_temp[index\_lvl\_prev+2\*i] = v[index\_lvl+i];

v\_temp[index\_lvl\_prev+2\*i+1] = 0.5\*(v[index\_lvl+i]+v[index\_lvl+i+1]);

}

return;

}

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 :

For :

1. **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 |