Multigrid grid solver for the pressure poisson equation

SANTHOSH RAO M

29 February 2020

Contents

List of Figures

1 Introduction

Multi-grid method is used to solve the given third order partial differential equation and the results are compared with the exact solution. This document contains the plots at various levels for the grid 32x32x32 and 16x16x16. In both the cases 4 levels have been used to compute. In-case of 32x32x32 for every twenty v-cycle once plots are made at each multi-grid level. Similarly for the 16x16x16 case for very ten v- cycle once, plots are made at each multi-grid level. Both the cases are solved for 0.000001 tolerance.

2 Results

2.1 32x32x32 Results

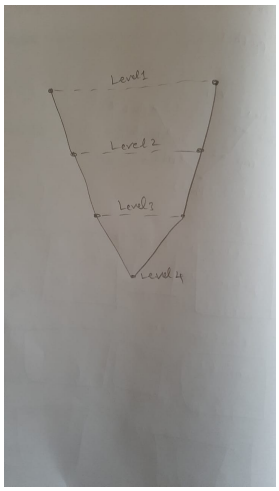
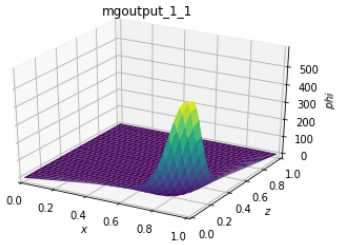
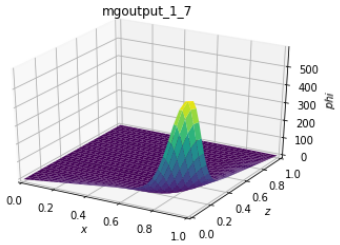


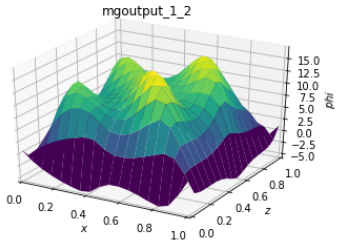
Figure 1: V-cycle-levels



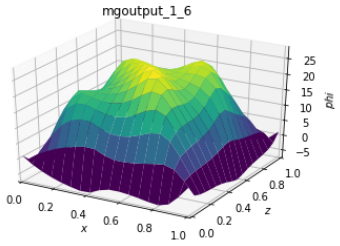
(a) Level-1



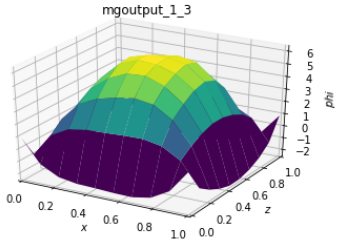
(b) Level-1



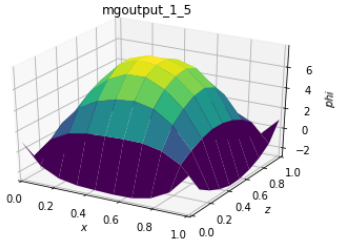
(c) Level-2



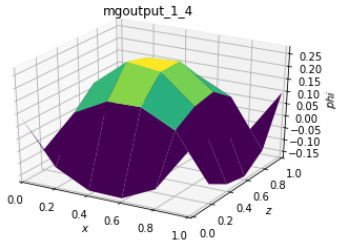
(d) Level-2



(e) Level-3

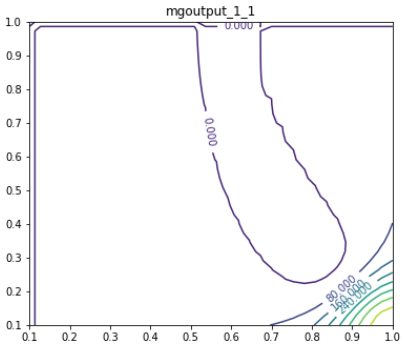


(f) Level-3

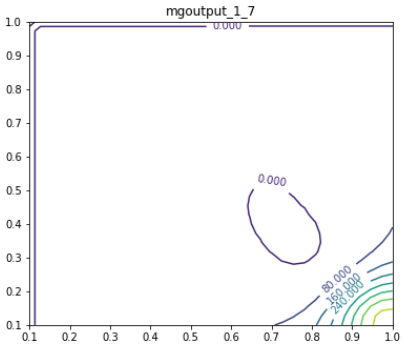


(g) Level-4

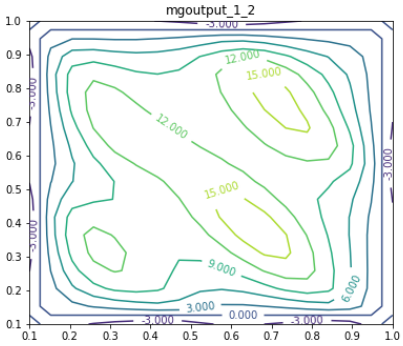
Figure 2: MG-Iteration-1



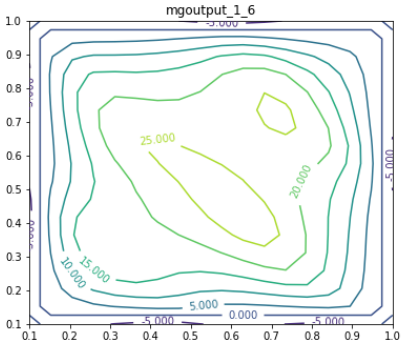
(a) Level-1



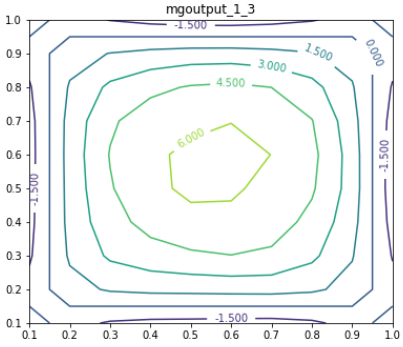
(b) Level-1



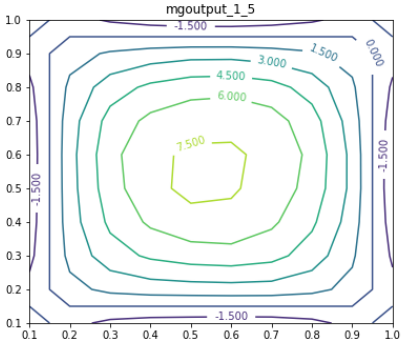
(c) Level-2



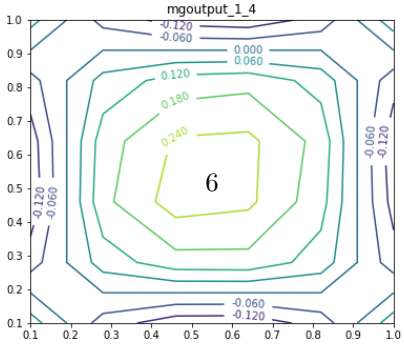
(d) Level-2



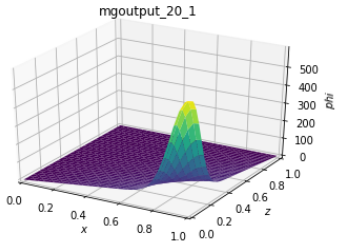
(e) Level-3



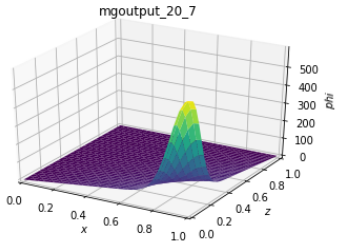
(f) Level-3



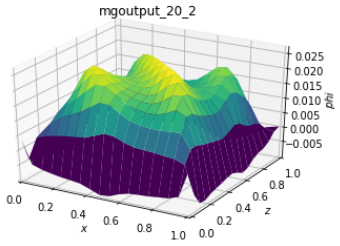
(g) Level-4



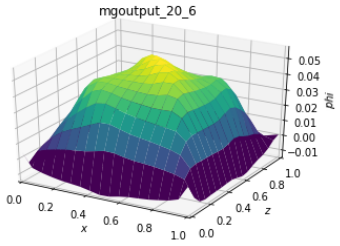
(a) Level-1



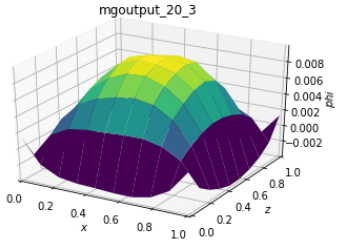
(b) Level-1



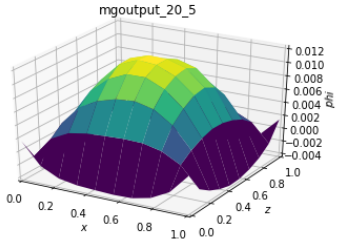
(c) Level-2



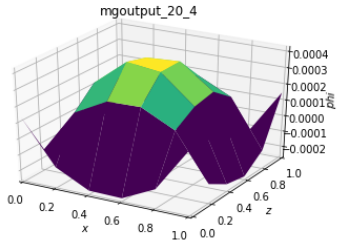
(d) Level-2



(e) Level-3

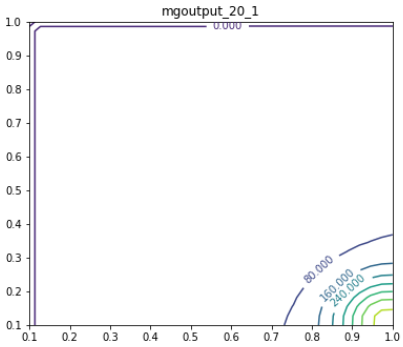


(f) Level-3

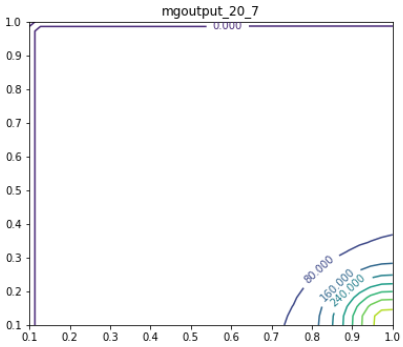


(g) Level-4

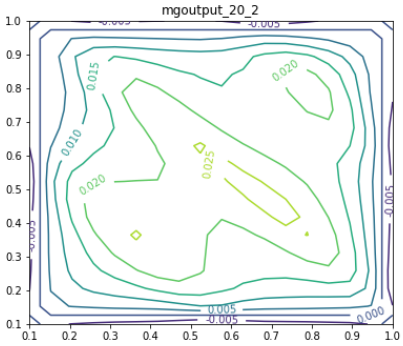
Figure 4: MG-Iteration-20



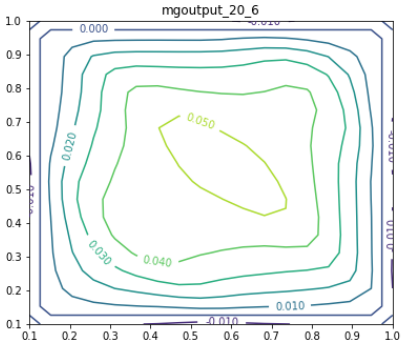
(a) Level-1



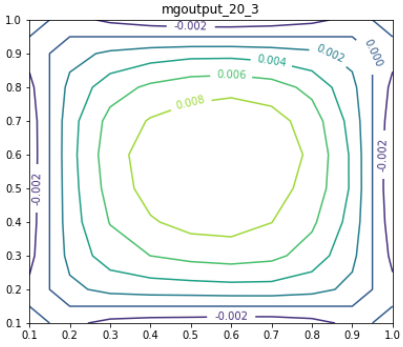
(b) Level-1



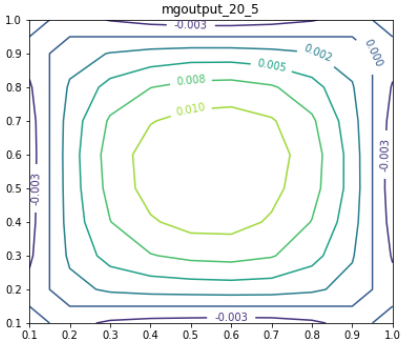
(c) Level-2



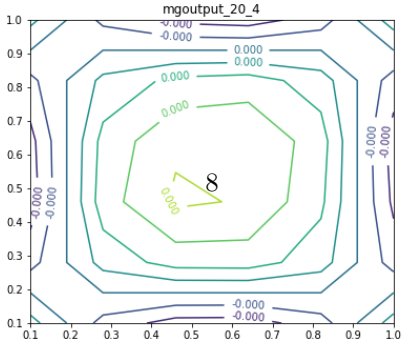
(d) Level-2



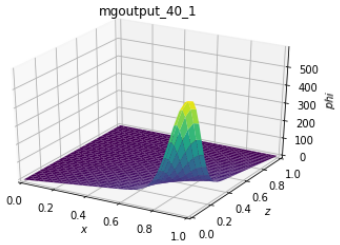
(e) Level-3



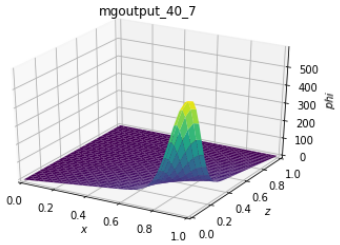
(f) Level-3



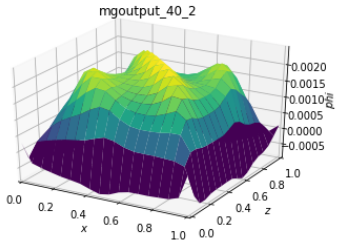
(g) Level-4



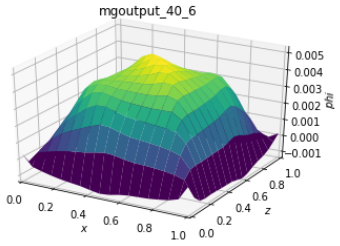
(a) Level-1



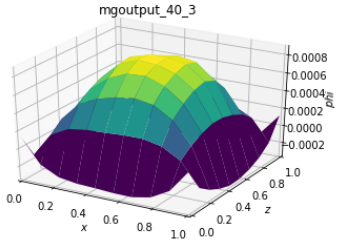
(b) Level-1



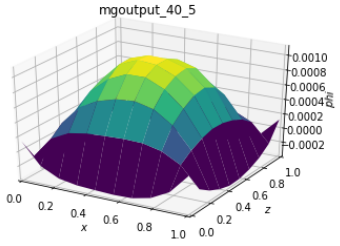
(c) Level-2



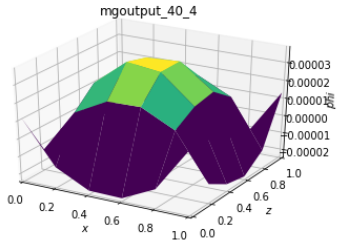
(d) Level-2



(e) Level-3

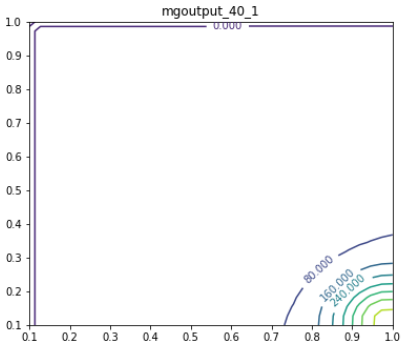


(f) Level-3

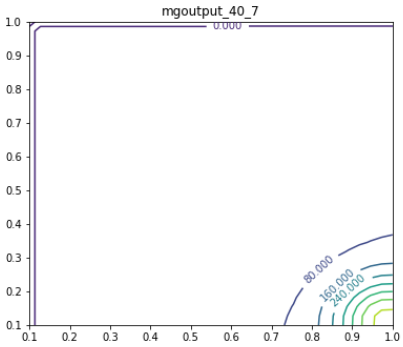


(g) Level-4

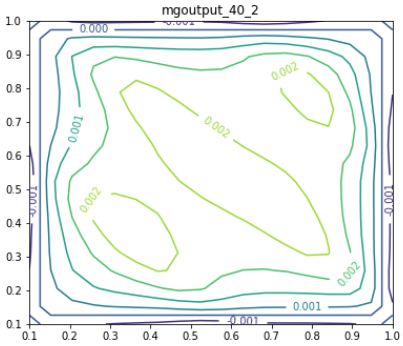
Figure 6: MG-Iteration-40



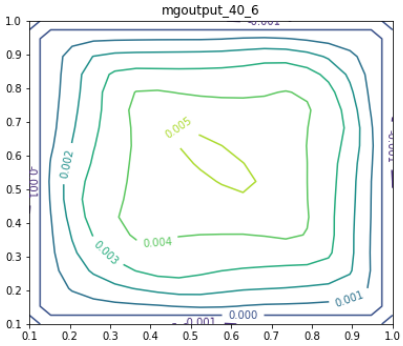
(a) Level-1



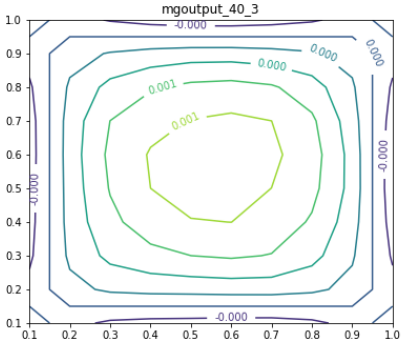
(b) Level-1



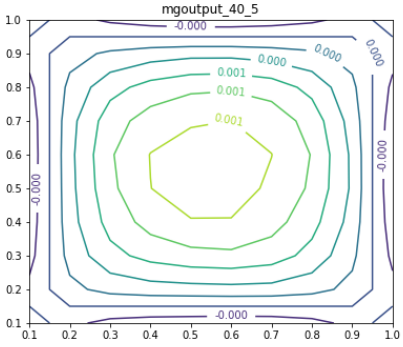
(c) Level-2



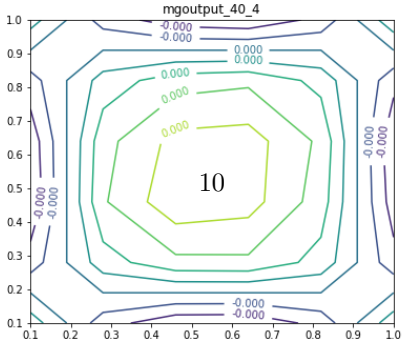
(d) Level-2



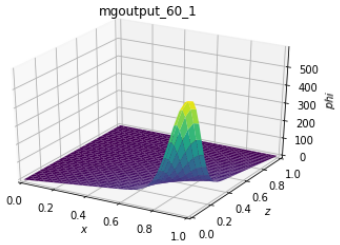
(e) Level-3



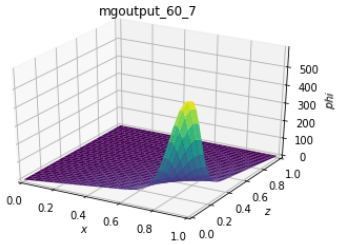
(f) Level-3



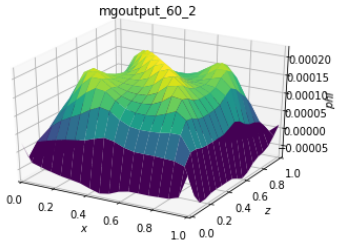
(g) Level-4



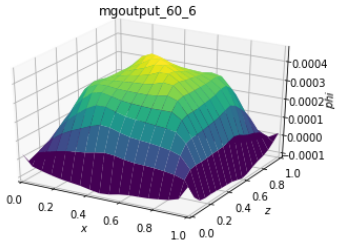
(a) Level-1



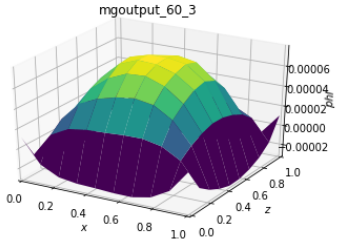
(b) Level-1



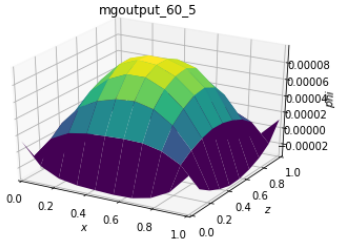
(c) Level-2



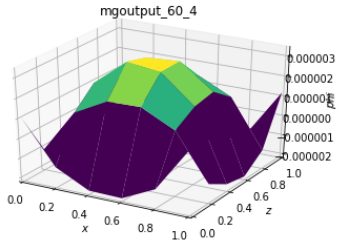
(d) Level-2



(e) Level-3

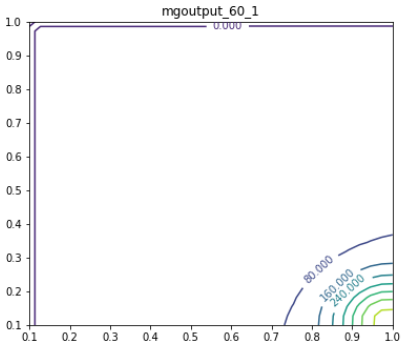


(f) Level-3

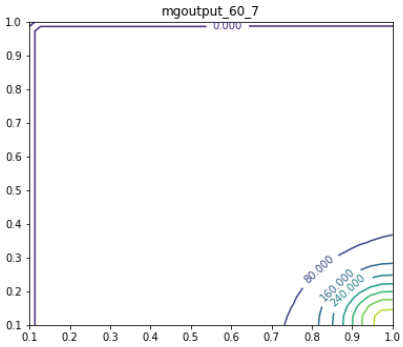


(g) Level-4

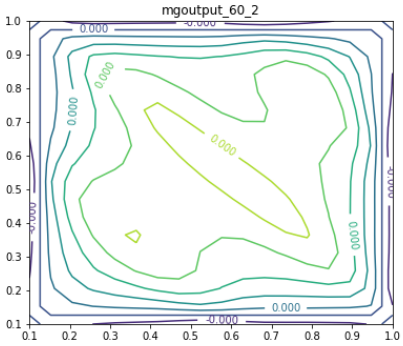
Figure 8: MG-Iteration-60



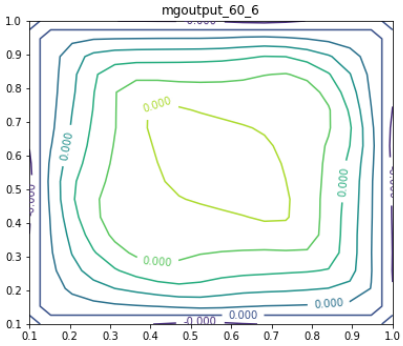
(a) Level-1



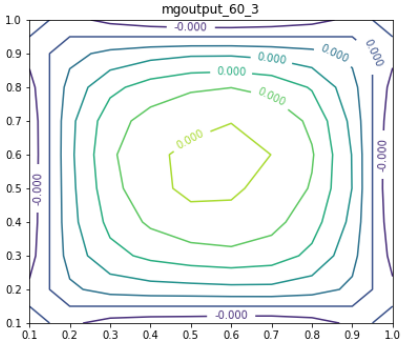
(b) Level-1



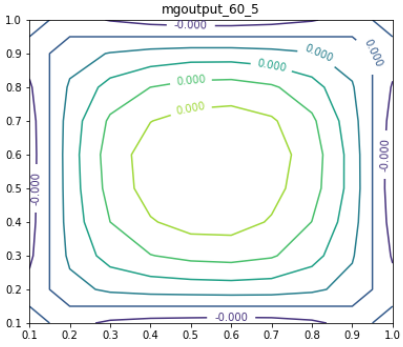
(c) Level-2



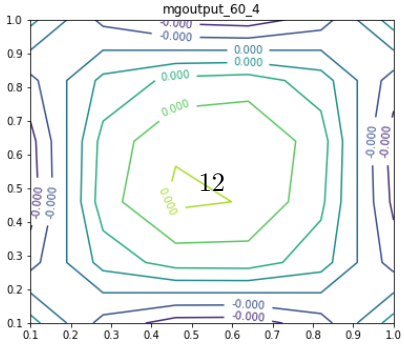
(d) Level-2



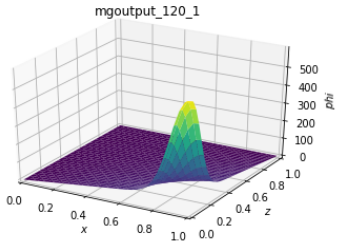
(e) Level-3



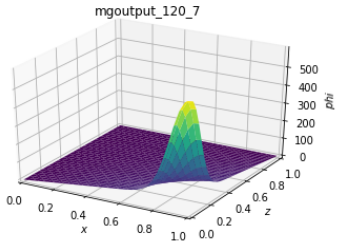
(f) Level-3



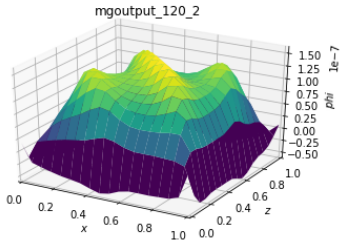
(g) Level-4



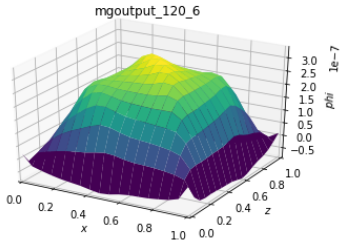
(a) Level-1



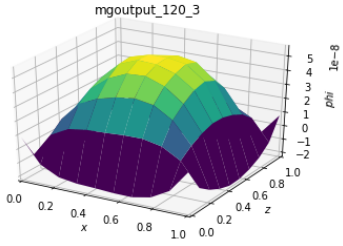
(b) Level-1



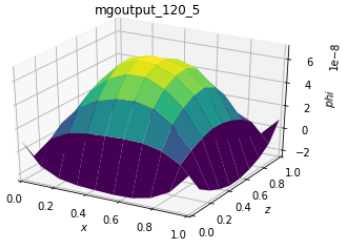
(c) Level-2



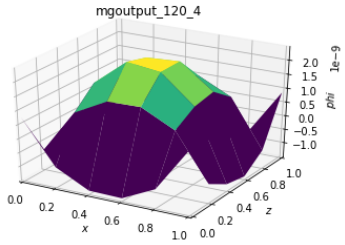
(d) Level-2



(e) Level-3

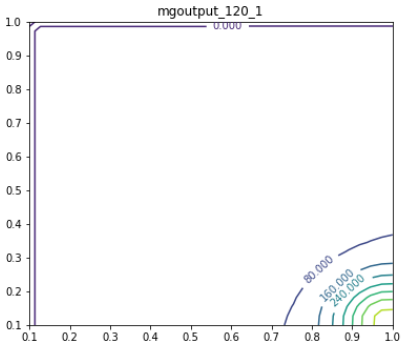


(f) Level-3

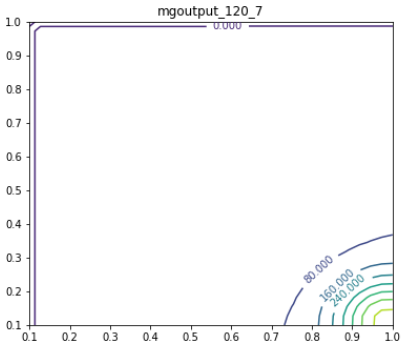


(g) Level-4

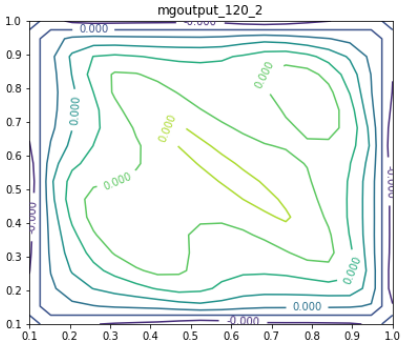
Figure 10: MG-Iteration-120



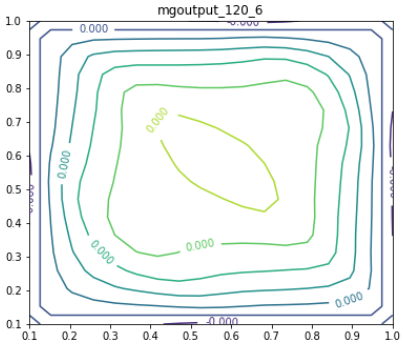
(a) Level-1



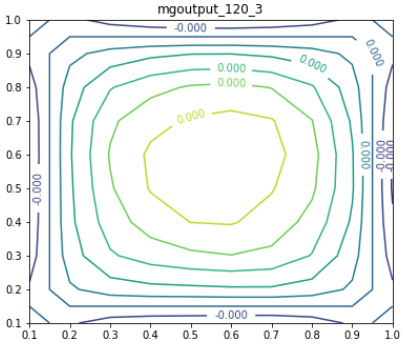
(b) Level-1



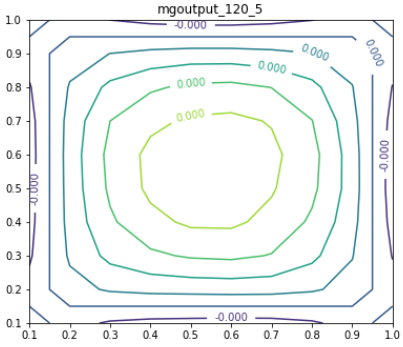
(c) Level-2



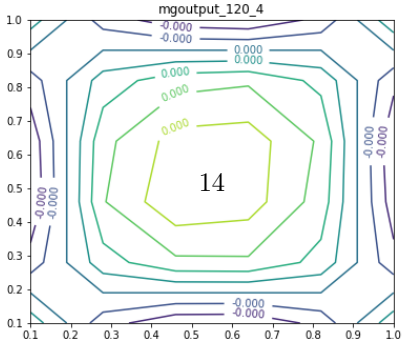
(d) Level-2



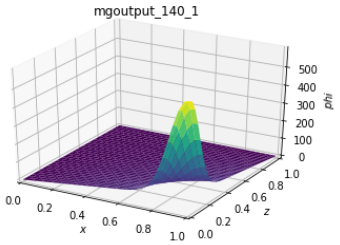
(e) Level-3



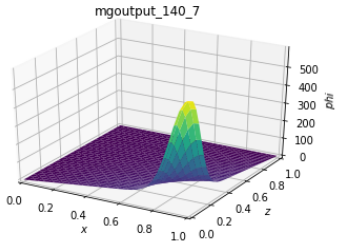
(f) Level-3



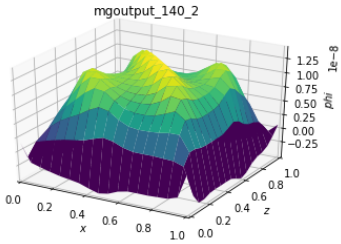
(g) Level-4



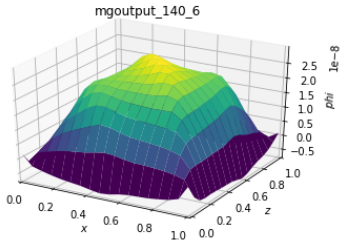
(a) Level-1



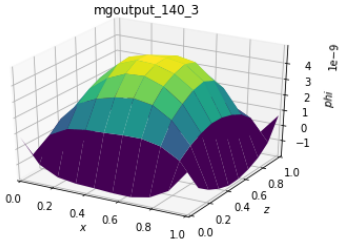
(b) Level-1



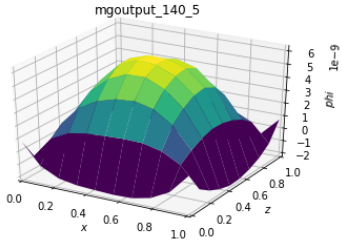
(c) Level-2



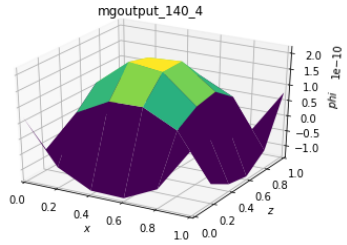
(d) Level-2



(e) Level-3

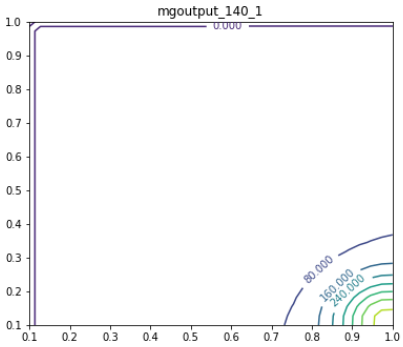


(f) Level-3

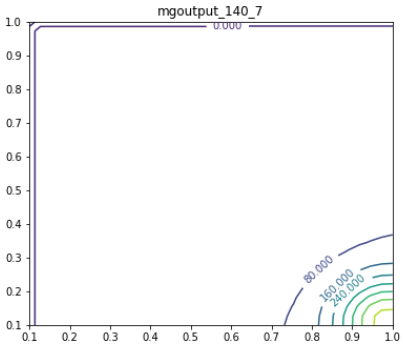


(g) Level-4

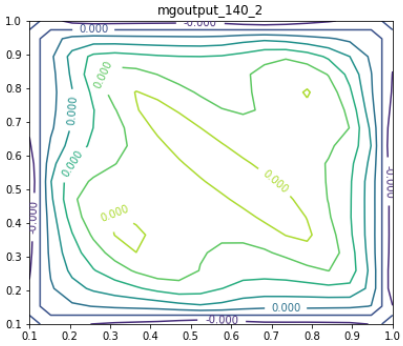
Figure 12: MG-Iteration-140



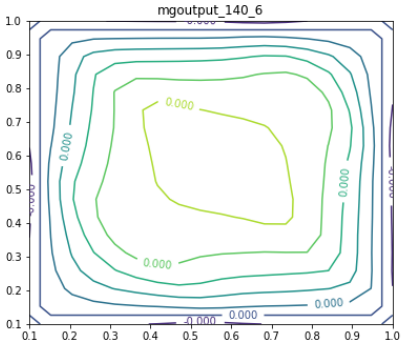
(a) Level-1



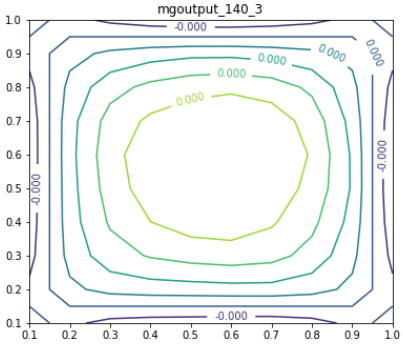
(b) Level-1



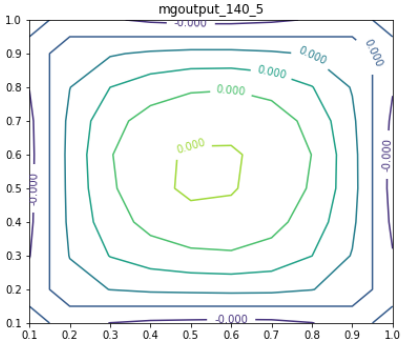
(c) Level-2



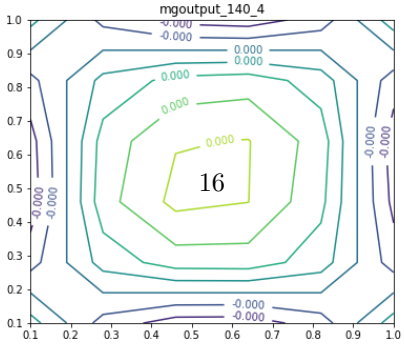
(d) Level-2



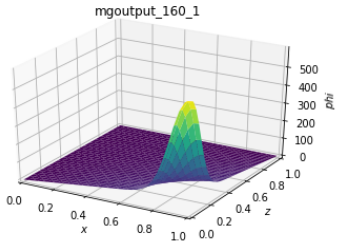
(e) Level-3



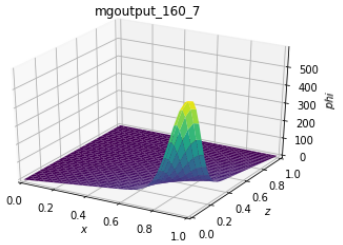
(f) Level-3



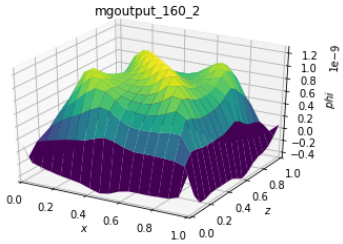
(g) Level-4



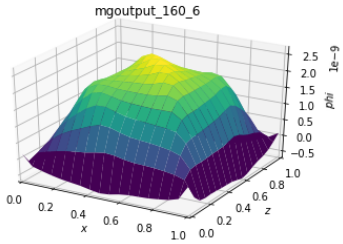
(a) Level-1



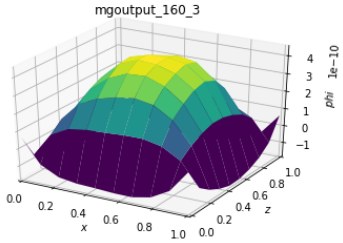
(b) Level-1



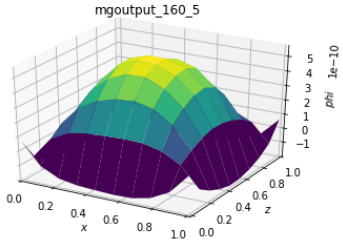
(c) Level-2



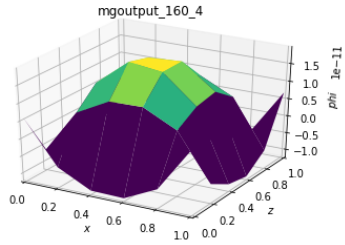
(d) Level-2



(e) Level-3

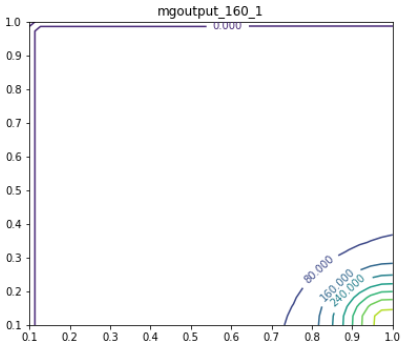


(f) Level-3

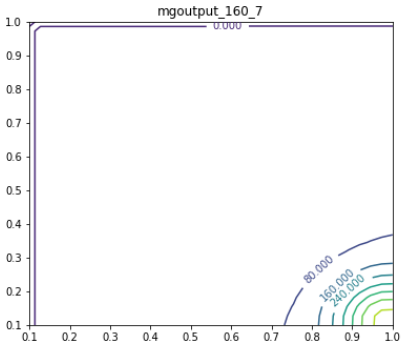


(g) Level-4

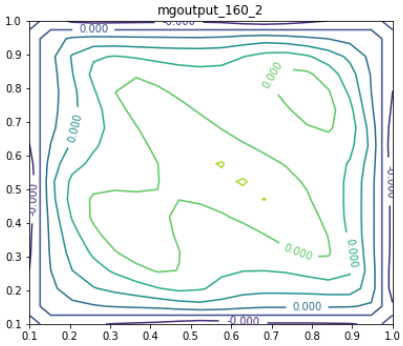
Figure 14: MG-Iteration-160



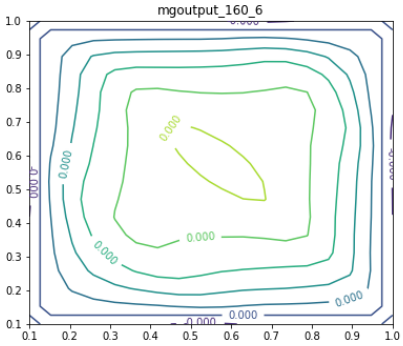
(a) Level-1



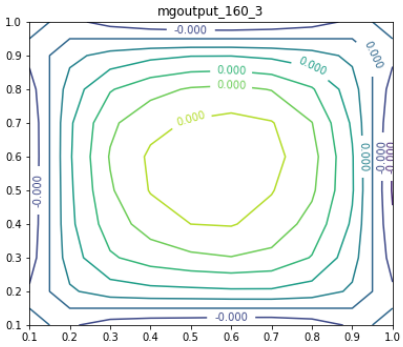
(b) Level-1



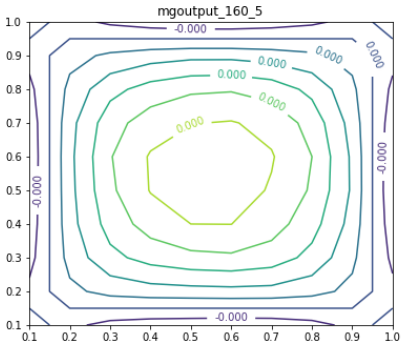
(c) Level-2



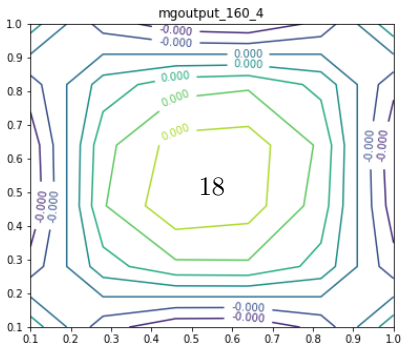
(d) Level-2



(e) Level-3

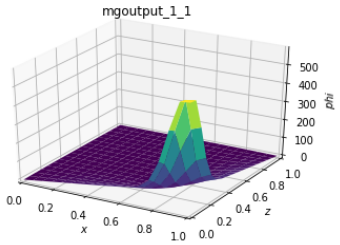


(f) Level-3

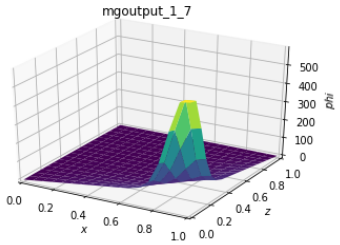


(g) Level-4

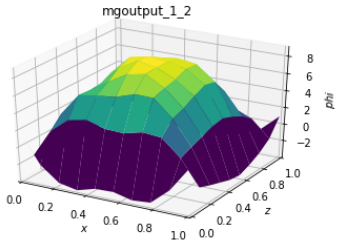
2.2 16x16x16 Results



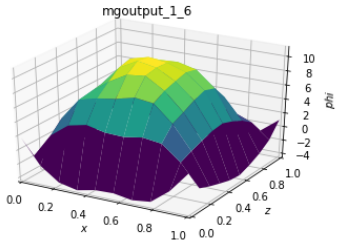
(a) Level-1



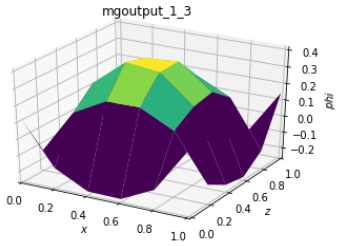
(b) Level-1



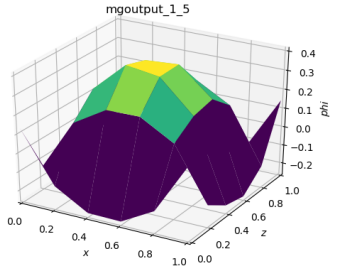
(c) Level-2



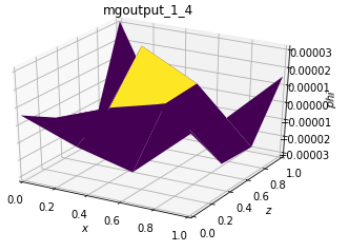
(d) Level-2



(e) Level-3

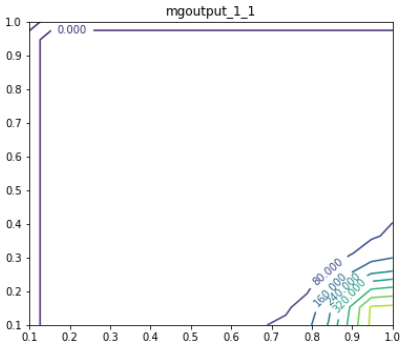


(f) Level-3

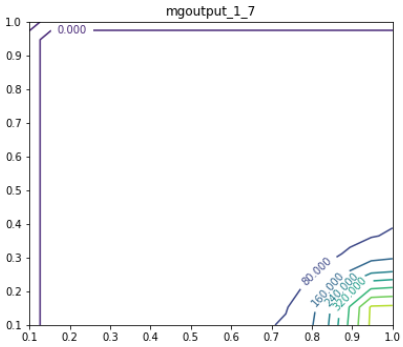


(g) Level-4

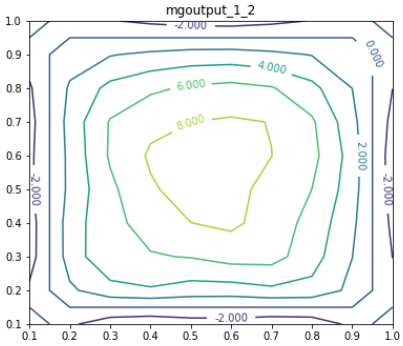
Figure 16: MG-Iteration-1



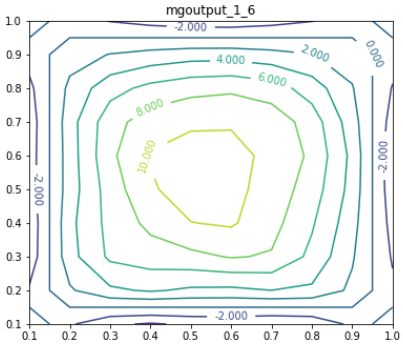
(a) Level-1



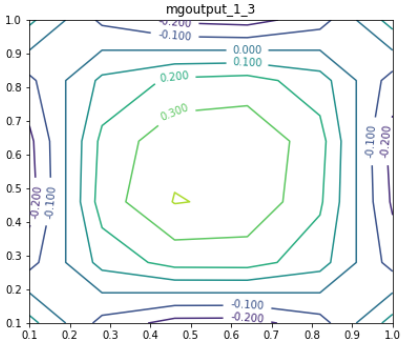
(b) Level-1



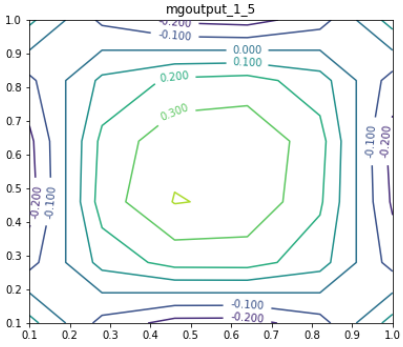
(c) Level-2



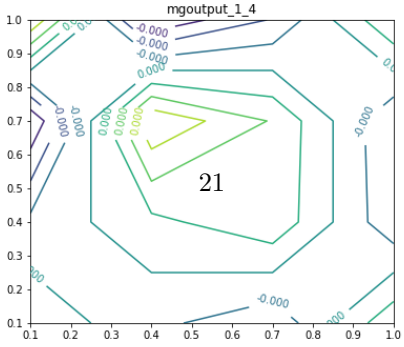
(d) Level-2



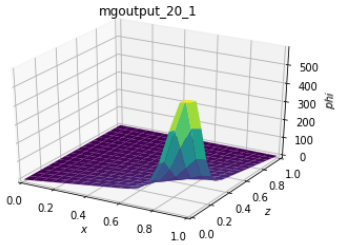
(e) Level-3



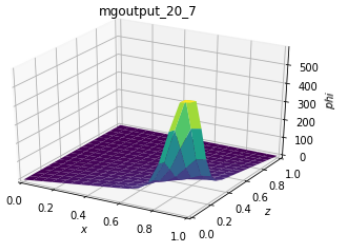
(f) Level-3



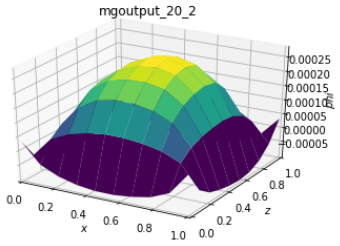
(g) Level-4



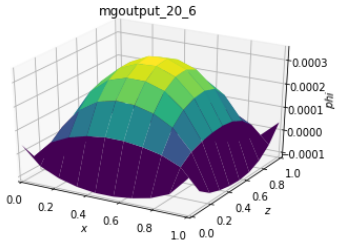
(a) Level-1



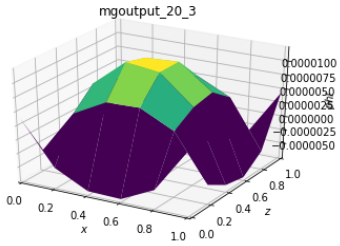
(b) Level-1



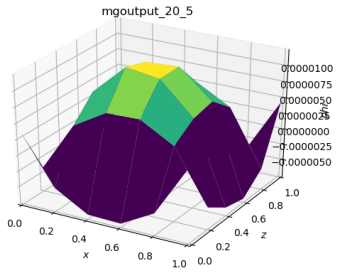
(c) Level-2



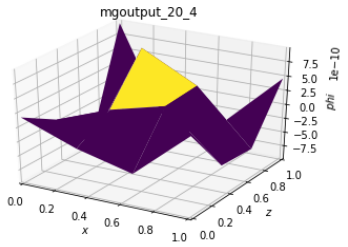
(d) Level-2



(e) Level-3

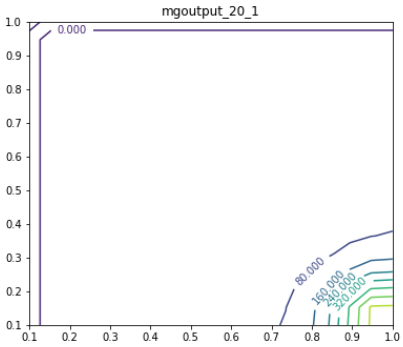


(f) Level-3

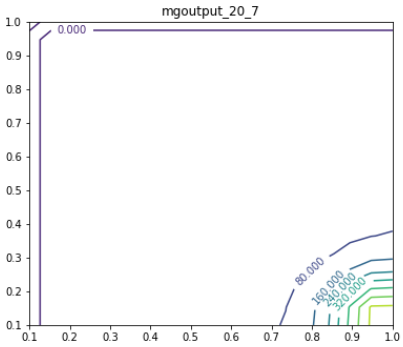


(g) Level-4

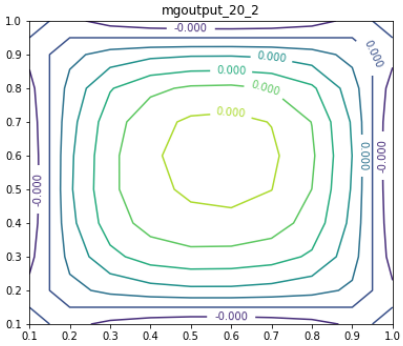
Figure 18: MG-Iteration-20



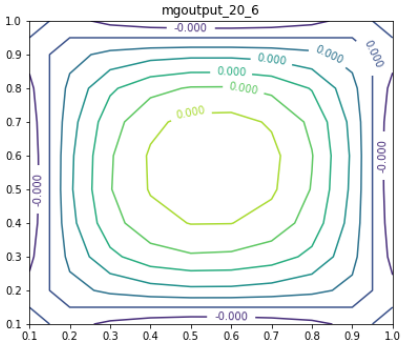
(a) Level-1



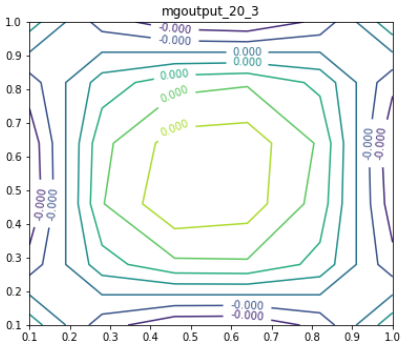
(b) Level-1



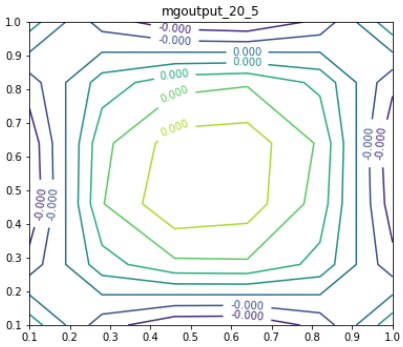
(c) Level-2



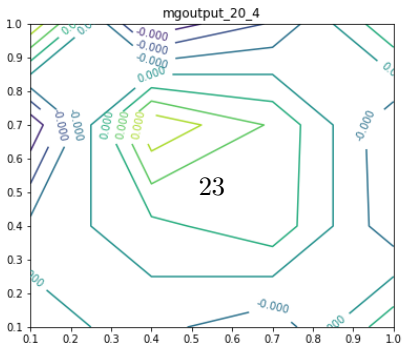
(d) Level-2



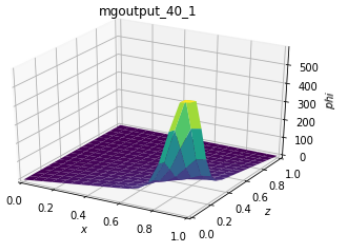
(e) Level-3



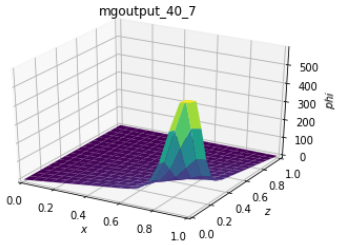
(f) Level-3



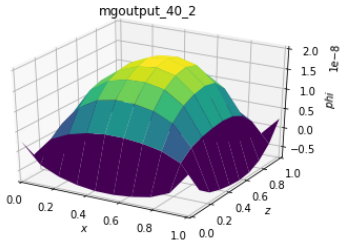
(g) Level-4



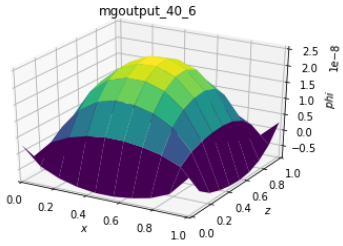
(a) Level-1



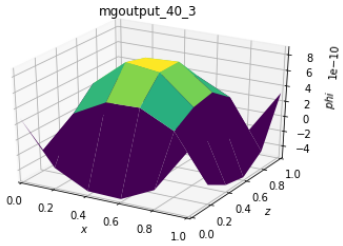
(b) Level-1



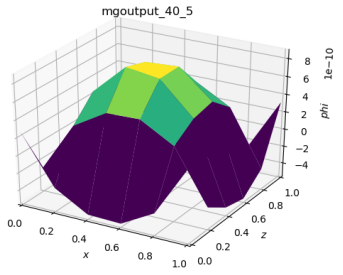
(c) Level-2



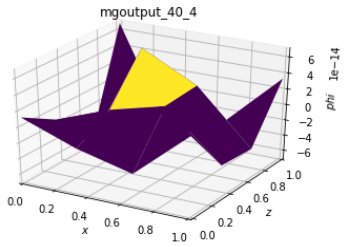
(d) Level-2



(e) Level-3

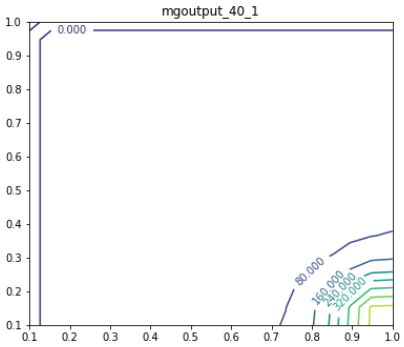


(f) Level-3

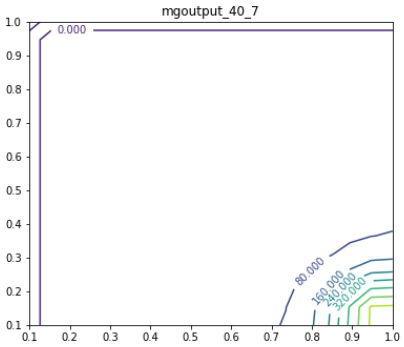


(g) Level-4

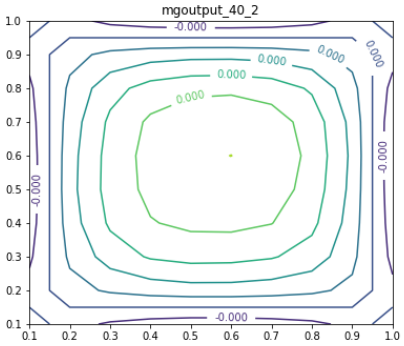
Figure 20: MG-Iteration-40



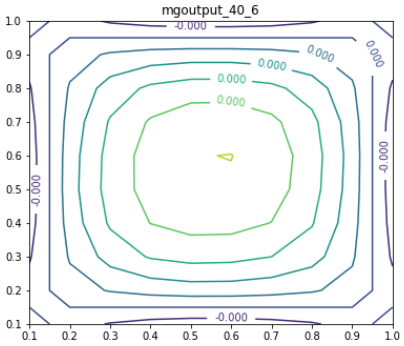
(a) Level-1



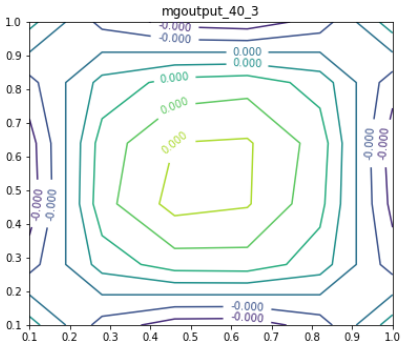
(b) Level-1



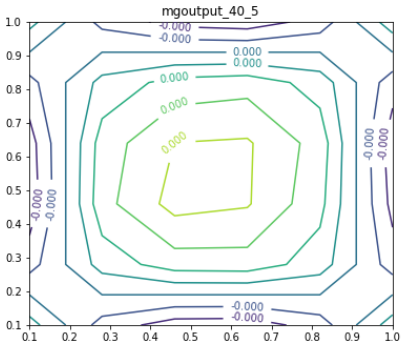
(c) Level-2



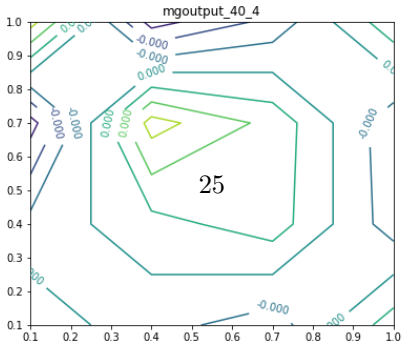
(d) Level-2



(e) Level-3



(f) Level-3



(g) Level-4

3 C++ Code

This code is used for generating the results. Change the nx, ny, nz for the desired grid size in the code. Pseudocode is given in the Figure 22, 23, 24. And the c++ code is also given below.

```
#include<iostream>
#include<math.h>
#include<vector>
#include<fstream>

#define phi(j,k,i) phi[(j*nxp2*nzp2)+(k*nxp2)+i]
#define phi_f(j,k,i) phi_f[(j*nxp2*nzp2)+(k*nxp2)+i]
#define phi_c(j,k,i) phi_c[(j*nxp2*nzp2)+(k*nxp2)+i]
#define phiold(j,k,i) phiold[(j*nxp2*nzp2)+(k*nxp2)+i]
#define phinew(j,k,i) phinew[(j*nxp2*nzp2)+(k*nxp2)+i]
#define fun(j,k,i) fun[(j*nxp2*nzp2)+(k*nxp2)+i]
#define rhs(j,k,i) rhs[(j*nxp2*nzp2)+(k*nxp2)+i]
#define res(j,k,i) res[(j*nxp2*nzp2)+(k*nxp2)+i]
#define newres(j,k,i) newres[(j*nxp2*nzp2)+(k*nxp2)+i]
#define oldres(j,k,i) oldres[(j*nxp2*nzp2)+(k*nxp2)+i]

using namespace std;


double rh(double x, double y, double z)
{
    double f_x = 50000.0 * exp(-50.0*(((1-x)*(1-x))+(z*z))) *
        (100*(((1-x)*(1-x))+(z*z))-2.0);
    return f_x;
}

void calculatexyz(vector<double> &x, vector<double> &y, vector<double>
    &z, int nx, int ny, int nz, double dxc, double dyc, double dzc)
{
    x.at(0) = -dxc/2.0, y.at(0) = -dyc/2.0, z.at(0) = -dzc/2.0;
    x[nx+1] = 1.0 + (dxc/2.0), y[ny+1] = 1.0 + (dyc/2.0), z[nz+1] = 1.0
        + (dzc/2.0);
    for (int i = 1; i < nx+1; i++)
    {
        x.at(i) = x.at(i-1)+dxc;
        y.at(i) = y.at(i-1)+dyc;
        z.at(i) = z.at(i-1)+dzc;
    }
}

void calculateFun(vector<double> &rhs, vector<double> &x, vector<double>
    &y, vector<double> &z, int nxp2, int nyp2, int nzp2, double dxc,
    double dyc, double dzc)
{
```

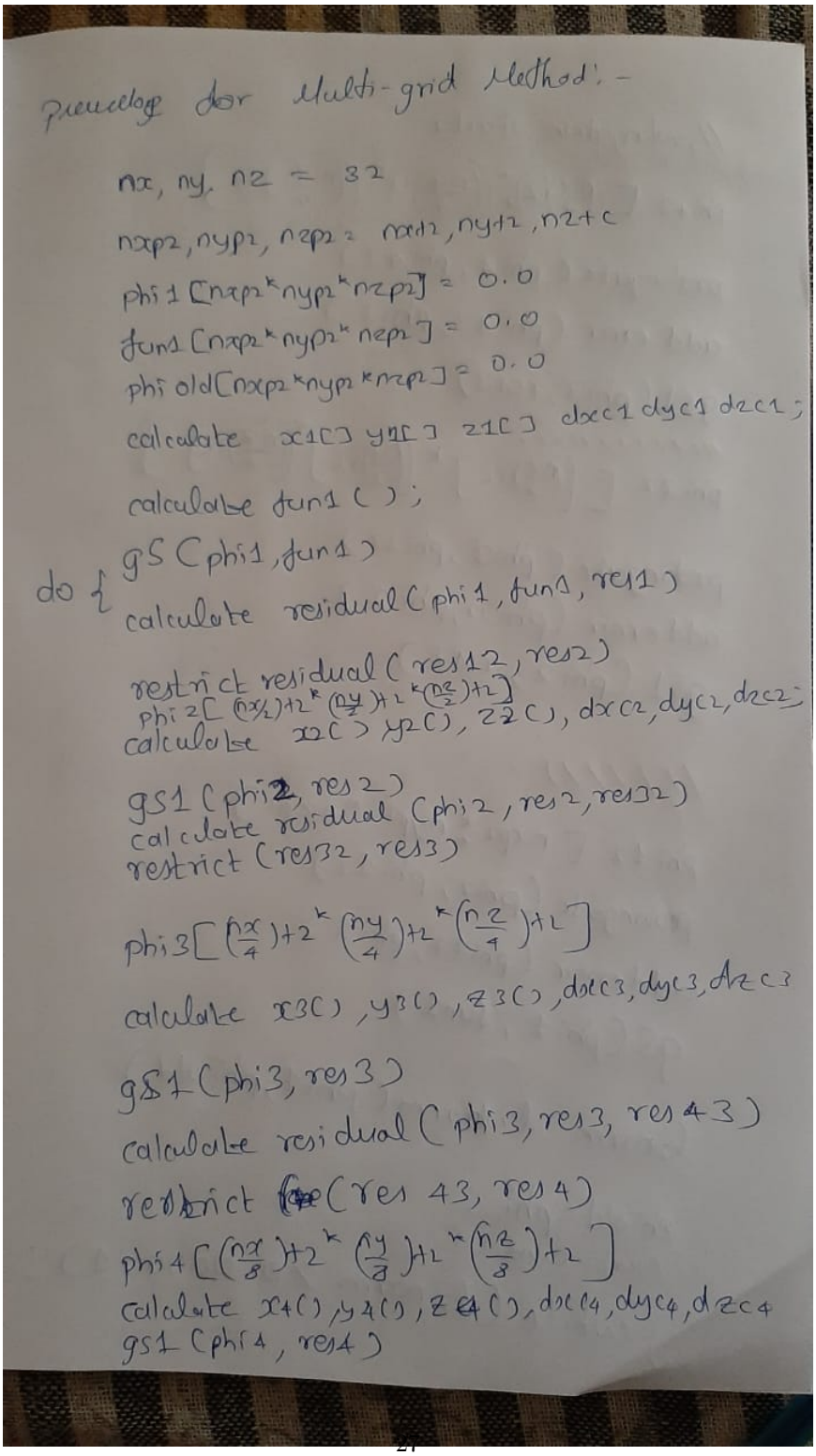


Figure 22: Pseudocode-1

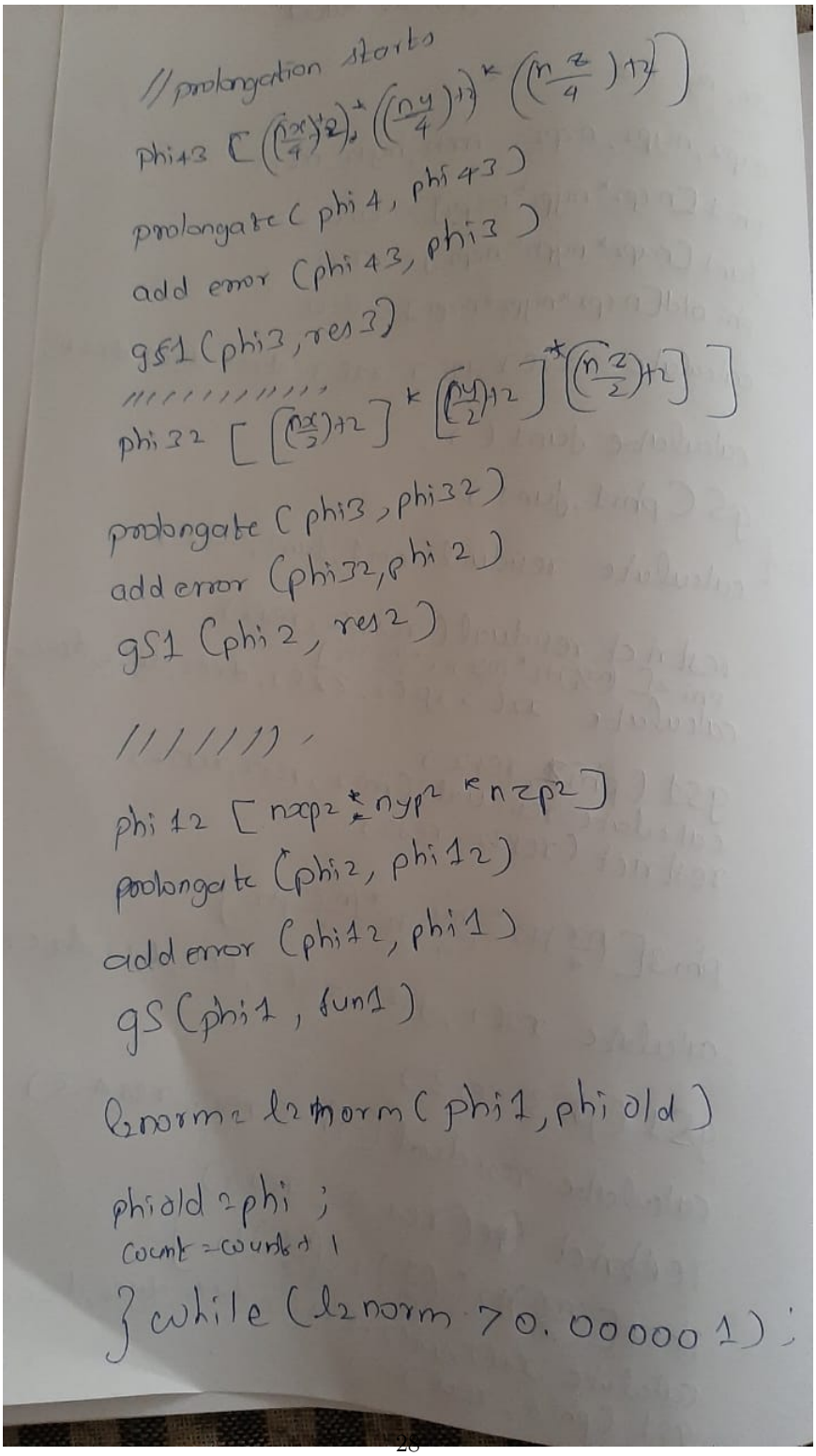


Figure 23: Pseudocode-2

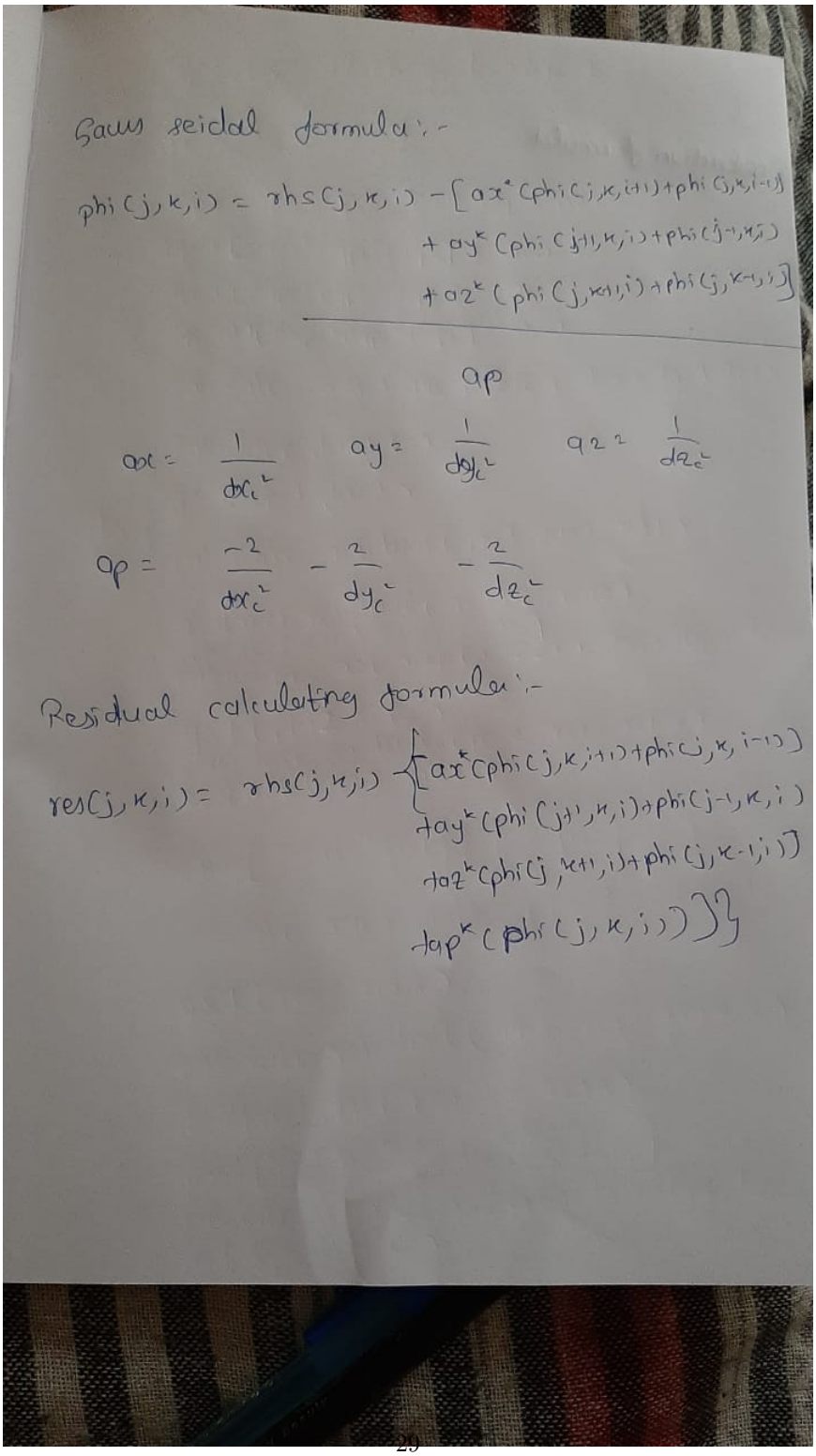


Figure 24: Pseudocode-3

```
    for(int j = 0; j < nyp2; j++)
        for (int k = 0; k < nzp2; k++)
            for (int i = 0; i < nxp2; i++)
                rhs(j,k,i) = rh(x[i],y[j],z[k]);
}
void boundaryValues(vector<double> &phi, vector<double> &x,
     vector<double> &y, vector<double> &z, int nxp2, int nyp2, int nzp2)
{
    for (int j = 0; j < nyp2; j++)
    {
        for (int i = 0; i < nxp2; i++)
        {
            phi(j,(0),i) =
                 2.0*((100.0*x[i])+(500.0*exp(-50.0*((1.0-x[i])*(1.0-x[i])))))
                 - phi(j,(1),i);

        }
    }
    for (int j = 0; j < nyp2; j++)
    {
        for (int i = 0; i < nxp2; i++)
        {
            phi(j,(nzp2-1),i) =
                 2.0*(500.0*exp(-50.0*(((1.0-x[i])*(1-x[i]))+1.0)))-phi(j,(nzp2-2),i);
        }
    }
    for (int j = 0; j < nyp2; j++)
    {
        for (int k = 0; k < nzp2; k++)
        {
            phi(j,k,(0)) = (2.0*(500.0*exp(-50.0*(1.0+(z[k]*z[k]))))) -
                 phi(j,k,(1));

        }
    }
    for (int j = 0; j < nyp2; j++)
    {
        for (int k = 0; k < nzp2; k++)
        {
            phi(j,k,(nxp2-1)) =
                 (2.0*((100.0*(1.0-z[k]))+(500.0*exp(-50.0*z[k]*z[k]))))
                 - phi(j,k,(nxp2-2));

        }
    }
}

void gS(vector<double> &phi, vector<double> &rhs, vector<double> &x,
     vector<double> &y, vector<double> &z,int nx, int ny, int nz, double
     dxc, double dyc, double dzc)
```

```
{
    double ap = (-2.0/(dxc*dxc))+(-2.0/(dyc*dyc))+(-2.0/(dzc*dzc));
    double ax = (1.0/(dxc*dxc)), ay = (1.0/(dyc*dyc)), az =
        (1.0/(dzc*dzc));
    int count = 0, nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
    do
    {
        for (int j = 1; j <(nyp2-1); j++)
        {
            for (int k = 1; k < (nzp2-1); k++)
            {
                for (int i = 1; i < (nxp2-1); i++)
                {
                    phi(j,k,i) = ((rhs(j,k,i) -
                        ((ax*(phi(j,k,(i+1))+phi(j,k,(i-1)))) \
                                                +
                                                    (ay*(phi((j+1),k,i)+phi((j-1),k,i)))
                                                    \
                                                +
                                                    (az*(phi(j,(k+1),i)+phi(j,(k-1),i))))))/ap;
                }
            }
      }
//Boundary conditions
    for (int j = 0; j < nyp2; j++)
    {
        for (int i = 0; i < nxp2; i++)
        {
            phi(j,(0),i) =
                2.0*((100.0*x[i])+(500.0*exp(-50.0*((1.0-x[i])*(1.0-x[i])))))
                - phi(j,(1),i);

        }
    }
    for (int j = 0; j < nyp2; j++)
    {
        for (int i = 0; i < nyp2; i++)
        {
            phi(j,(nzp2-1),i) =
                2.0*(500.0*exp(-50.0*(((1.0-x[i])*(1-x[i]))+1.0)))-phi(j,(nzp2-2),i);
        }
    }
    for (int j = 0; j < nyp2; j++)
    {
        for (int k = 0; k < nzp2; k++)
        {
            phi(j,k,(0)) = (2.0*(500.0*exp(-50.0*(1.0+(z[k]*z[k])))) -
                phi(j,k,(1));

        }
```

```
    }
    for (int j = 0; j < nyp2; j++)
    {
        for (int k = 0; k < nzp2; k++)
        {
            phi(j,k,(nxp2-1)) =
                (2.0*((100.0*(1.0-z[k]))+(500.0*exp(-50.0*z[k]*z[k]))))
                - phi(j,k,(nxp2-2));
        }
    }
    for (int j = 0; j < nyp2; j++)
        {
            for (int k = 0; k < nzp2; k++)
            {
                for (int i = 0; i < nxp2; i++)
                {
                    if(j==0)
                    {
                        phi(j,k,i) = phi((nyp2-1),k,i);
                    }
                    if(j==nyp2-2)
                    {
                        phi(j,k,i) = phi((1),k,i);
                    }
                }
            }
        }
        count = count+1;
    }while (count<12);
}

/*void printLevel( vector<double> phi, int nxp2, int nyp2, int nzp2, int
    level)
{
    ofstream gs("mgoutput.txt",ios::trunc);
    for (int k = 0; k < nzp2; k++)
        {
            for (int i = 0; i < nxp2; i++)
            {
                //cout << var->p[i][0][k] << endl;
                //cout << "p[" << level<<"]"<<"["<<k<<"]"<<"["<<i<<"]: "
                    << var->p[level][k][i] << " ";
                gs << phi(level,k,i) <<",";

            }
            gs << endl;
        }
        //cout << endl;
    gs.close();
}
```

```
*/
void gS1(vector<double> &phi, vector<double> &rhs, vector<double> &x,
    vector<double> &y, vector<double> &z,int nx, int ny, int nz, double
    dxc, double dyc, double dzc)
{
   double ap = (-2.0/(dxc*dxc))+(-2.0/(dyc*dyc))+(-2.0/(dzc*dzc));
   double ax = (1.0/(dxc*dxc)), ay = (1.0/(dyc*dyc)), az =
       (1.0/(dzc*dzc));
   int count = 0, nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
   do
   {
      for (int j = 1; j <(nyp2-1); j++)
      {
         for (int k = 1; k < (nzp2-1); k++)
         {
            for (int i = 1; i < (nxp2-1); i++)
            {
               phi(j,k,i) = ((rhs(j,k,i) -
                   ((ax*(phi(j,k,(i+1))+phi(j,k,(i-1)))) \
                                          +
                                              (ay*(phi((j+1),k,i)+phi((j-1),k,i)))
                                              \
                                           +
                                               (az*(phi(j,(k+1),i)+phi(j,(k-1),i))))))/ap;
            }
         }
      }
//Boundary conditions
   for (int j = 0; j < nyp2; j++)
   {
      for (int i = 0; i < nxp2; i++)
      {
         phi(j,(0),i) = - phi(j,(1),i);

      }
   }
   for (int j = 0; j < nyp2; j++)
   {
      for (int i = 0; i < nyp2; i++)
      {
         phi(j,(nzp2-1),i) = -phi(j,(nzp2-2),i);
      }
   }
   for (int j = 0; j < nyp2; j++)
   {
      for (int k = 0; k < nzp2; k++)
      {
         phi(j,k,(0)) = - phi(j,k,(1));
```

```
        }
    }
    for (int j = 0; j < nyp2; j++)
    {
        for (int k = 0; k < nzp2; k++)
        {
            phi(j,k,(nxp2-1)) = - phi(j,k,(nxp2-2));
        }
    }
//y- direction
    for (int k = 0; k < nzp2; k++)
    {
        for (int i = 0; i < nxp2; i++)
        {
            phi((0),k,i) = - phi((1),k,i);

        }
    }
    for (int k = 0; k < nzp2; k++)
    {
        for (int i = 0; i < nzp2; i++)
        {
            phi((nyp2-1),k,i) = - phi((nyp2-2),k,i);
        }
    }

    /*for (int j = 0; j < nyp2; j++)
        {
            for (int k = 0; k < nzp2; k++)
            {
                for (int i = 0; i < nxp2; i++)
                {
                    if(j==0)
                    {
                        phi(j,k,i) = phi((nyp2-1),k,i);
                    }
                    if(j==nyp2-2)
                    {
                        phi(j,k,i) = phi((1),k,i);
                    }
                }
            }
        }*/


        count = count+1;
    }while (count<12);
}

void printLevel( vector<double> phi, int nx, int ny, int nz, int level,
```

```
    int count, int levelno)
{
    int nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
    string init("mgoutput_");
    string add(to_string(count));

    // Appending the string.
    init.append(add);
    init.append("_");
    string levelnumber = to_string(levelno);
    init.append(levelnumber);

    ofstream gs(init,ios::trunc);
    for (int k = 0; k < nzp2; k++)
        {
            for (int i = 0; i < nxp2; i++)
            {
                //cout << var->p[i][0][k] << endl;
                //cout << "p[" << level<<"]"<<"["<<k<<"]"<<"["<<i<<"]: "
                    << var->p[level][k][i] << " ";
                gs << phi(level,k,i) <<",";

            }
            gs << endl;
        }
        //cout << endl;
    gs.close();
}
void printAll( vector<double> fun, int nxp2, int nyp2, int nzp2)
{
    for (int j = 0; j < nyp2; j++)
    {
        for (int k = 0; k < nzp2; k++)
        {
            for (int i = 0; i < nxp2; i++)
            {
                cout << "fun[" << j<<"]"<<"["<<k<<"]"<<"["<<i<<"]: " <<
                    fun(j,k,i) << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
}
void calResidual(vector<double> &phi,vector<double> &rhs, vector<double>
    &res, int nx, int ny, int nz, double dxc, double dyc, double dzc)
{
    int nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
    double ap = (-2.0/(dxc*dxc))+(-2.0/(dyc*dyc))+(-2.0/(dzc*dzc));
    double ax = (1.0/(dxc*dxc)), ay = (1.0/(dyc*dyc)), az =
```

```
        (1.0/(dzc*dzc));
    //vector<double> res;
    //res.assign(nxp2*nyp2*nzp2, 0.0);
    for (int j = 1; j < (nyp2-1); j++)
        for (int k = 1; k < (nzp2-1); k++)
            for (int i = 1; i < (nxp2-1); i++)
               {
                res(j,k,i) = rhs(j,k,i) -
                    ((ax*(phi(j,k,(i+1))+phi(j,k,(i-1)))) \
                                            +
                                                (ay*(phi((j+1),k,i)+phi((j-1),k,i)))
                                                \
                                            +
                                                (az*(phi(j,(k+1),i)+phi(j,(k-1),i)))+(ap*phi(j,k,i)));
               }

    //return res;

}
void restrict(vector<double> &oldres, vector<double> &newres, int nx,
    int ny, int nz)
{
    int nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
    for (int j = 1; j < ((nyp2-1)); j++)
        for (int k = 1; k < (nzp2-1); k++)
            for (int i = 1; i < (nxp2-1); i++)
                {
                    newres(j,k,i) = 0.125*(
                        oldres((2*j-1),(2*k-1),(2*i-1))\
                                    +oldres((2*j),(2*k-1),(2*i-1))\
                                    +oldres((2*j-1),(2*k),(2*i-1))\
                                    +oldres((2*j),(2*k),(2*i-1))\
                                    +oldres((2*j-1),(2*k-1),(2*i))
                                    +oldres((2*j),(2*k-1),(2*i))\
                                    +oldres((2*j-1),(2*k),(2*i))\
                                    +oldres((2*j),(2*k),(2*i)));
                }
}
void prolongate(vector<double> &phi_c, vector<double> &phi_f, int nx,
    int ny, int nz)
{
    int nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
    double pa = 27.0/64.0, pb = 9.0/64.0, pc = 3.0/64.0, pd = 1.0/64.0;
    for (int j = 1; j < nyp2-1; j++)
        for (int k = 1; k < nzp2-1; k++)
            for (int i = 1; i < nxp2-1; i++)
            {
                phi_f((2*j-1),(2*k-1),(2*i-1)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j-1),k,i) + phi_c(j,(k-1),i) +
                    phi_c(j,k,(i-1))) + pc*(phi_c((j-1),(k-1),i) +
```

```
                    phi_c((j-1),k,(i-1)) + phi_c(j,(k-1),(i-1))) +
                    pd*phi_c((j-1),(k-1),(i-1));
                phi_f((2*j),(2*k-1),(2*i-1)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j+1),k,i) + phi_c(j,(k-1),i) +
                    phi_c(j,k,(i-1))) + pc*(phi_c((j+1),(k-1),i) +
                    phi_c((j+1),k,(i-1)) + phi_c(j,(k-1),(i-1))) +
                    pd*phi_c((j+1),(k-1),(i-1));
                phi_f((2*j-1),(2*k),(2*i-1)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j-1),k,i) + phi_c(j,(k+1),i) +
                    phi_c(j,k,(i-1))) + pc*(phi_c((j-1),(k+1),i) +
                    phi_c((j-1),k,(i-1)) + phi_c(j,(k+1),(i-1))) +
                    pd*phi_c((j-1),(k+1),(i-1));
                phi_f((2*j),(2*k),(2*i-1)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j+1),k,i) + phi_c(j,(k+1),i) +
                    phi_c(j,k,(i-1))) + pc*(phi_c((j+1),(k+1),i) +
                    phi_c((j+1),k,(i-1)) + phi_c(j,(k+1),(i-1))) +
                    pd*phi_c((j+1),(k+1),(i-1));
                phi_f((2*j-1),(2*k-1),(2*i)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j-1),k,i) + phi_c(j,(k-1),i) +
                    phi_c(j,k,(i+1))) + pc*(phi_c((j-1),(k-1),i) +
                    phi_c((j-1),k,(i+1)) + phi_c(j,(k-1),(i+1))) +
                    pd*phi_c((j-1),(k-1),(i+1));
                phi_f((2*j),(2*k-1),(2*i)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j+1),k,i) + phi_c(j,(k-1),i) +
                    phi_c(j,k,(i+1))) + pc*(phi_c((j+1),(k-1),i) +
                    phi_c((j+1),k,(i+1)) + phi_c(j,(k-1),(i+1))) +
                    pd*phi_c((j+1),(k-1),(i+1));
                phi_f((2*j-1),(2*k),(2*i)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j-1),k,i) + phi_c(j,(k+1),i) +
                    phi_c(j,k,(i+1))) + pc*(phi_c((j-1),(k+1),i) +
                    phi_c((j-1),k,(i+1)) + phi_c(j,(k+1),(i+1))) +
                    pd*phi_c((j-1),(k+1),(i+1));
                phi_f((2*j),(2*k),(2*i)) = pa*phi_c(j,k,i) +
                    pb*(phi_c((j+1),k,i) + phi_c(j,(k+1),i) +
                    phi_c(j,k,(i+1))) + pc*(phi_c((j+1),(k+1),i) +
                    phi_c((j+1),k,(i+1)) + phi_c(j,(k+1),(i+1))) +
                    pd*phi_c((j+1),(k+1),(i+1));
            }
}
void adderror(vector<double> &phinew, vector<double> &phiold, int nx,
    int ny, int nz)
{
    int nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
    for (int j = 0; j < nyp2; j++)
        for (int k = 0; k < nzp2; k++)
            for (int i = 0; i < nxp2; i++)
            {
                phiold(j,k,i) = phiold(j,k,i) + phinew(j,k,i);

            }
```

```
}
double l2norm(vector<double> &phi, vector<double> &phiold, int nx, int
    ny, int nz)
{
       int nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;
       double temp = 0.0;
       double l2norm = 1.0;
       for (int j = 0; j < nyp2; j++)
       {
           for (int k = 0; k < nzp2; k++)
           {
               for (int i = 0; i < nxp2; i++)
               {
                   temp = temp + pow((phi(j,k,i)-phiold(j,k,i)),2.0);
               }

           }

       }
       l2norm = sqrt(temp);
       cout << l2norm << endl;
       return l2norm;
}
void copyphi(vector<double> &phi, vector<double> &phiold, int nx, int
    ny, int nz)
{
   int nxp2 = nx+2, nyp2 = ny+2, nzp2 = nz+2;

   for (int j = 0; j < nyp2; j++)
          for (int k = 0; k < nzp2; k++)
              for (int i = 0; i < nxp2; i++)
                  phiold(j,k,i) = phi(j,k,i);

}
int main()
{
   int nx=16 , ny = 16, nz = 16;
   int nxp2= nx+2, nyp2 = ny+2, nzp2 = nz+2;
   //int nxp2 = nx+1, nyp2 = ny+1, nzp2 = nz+1;
   double dxc1 = 1.0/(nx), dyc1 = 1.0/(ny), dzc1 = 1.0/(nz);
   vector<double> phi1, x1, y1, z1;
   vector<double> fun1;
   vector<double> phiold;
   phi1.assign(nxp2*nyp2*nzp2,0.0);
   fun1.assign(nxp2*nyp2*nzp2,0.0);
   phiold.assign(nxp2*nyp2*nzp2,0.0);
   x1.assign(nxp2,0.0);
   y1.assign(nyp2,0.0);
   z1.assign(nzp2,0.0);
   calculatexyz(x1,y1,z1,nx,ny,nz,dxc1,dyc1,dzc1);
```

```
  calculateFun(fun1, x1, y1, z1, nxp2,nyp2,nzp2,dxc1,dyc1,dzc1);
  gS(phi1, fun1, x1, y1, z1, nx, ny, nz, dxc1, dyc1, dzc1);

  //boundaryValues(phi1, x1, y1, z1, nxp2, nyp2, nzp2);
  double vl2norm=0.0;
  int count = 1;
  printLevel(phi1, nx, ny, nz , 8, count, 1);
  do
  {

  //printLevel(phi1,nxp2,nyp2,nzp2,31);
  //printAll(fun1,nxp2,nyp2,nzp2);
  vector<double> res12,res2;
  res12.assign((nxp2*nyp2*nzp2),0.0);
  calResidual(phi1, fun1, res12, nx, ny, nz, dxc1, dyc1, dzc1);
  //vector<double>().swap(fun1);
  res2.assign((((nx/2)+2)*((ny/2)+2)*((nz/2)+2)),0.0);
//  int n = res2.size();
  //cout << endl << n << endl;
  restrict(res12,res2, (nx/2), (ny/2), (nz/2));

  vector<double> phi2,x2,y2,z2;
  phi2.assign((((nx/2)+2)*((ny/2)+2)*((nz/2)+2)),0.0);
  x2.assign(((nx/2)+2),0.0);
  y2.assign(((ny/2)+2),0.0);
  z2.assign(((nz/2)+2),0.0);
  double dxc2 = (1.0)/(nx/2.0), dyc2 = (1.0)/(ny/2.0), dzc2 =
      (1.0)/(nz/2.0);
  calculatexyz(x2,y2,z2,(nx/2),(ny/2),(nz/2),dxc2,dyc2,dzc2);
  gS1(phi2, res2, x2, y2, z2, (nx/2), (ny/2), (nz/2), dxc2, dyc2,
      dzc2);

  if((count == 1) || (count%20==0) )
  printLevel(phi2, (nx/2), (ny/2), (nz/2), 4, count, 2);

  vector<double> res32, res3;
  res32.assign((((nx/2)+2)*((ny/2)+2)*((nz/2)+2)),0.0);
  calResidual(phi2,res2,res32, (nx/2), (ny/2), (nz/2), dxc2, dyc2,
      dzc2);
  res3.assign((((nx/4)+2)*((ny/4)+2)*((nz/4)+2)), 0.0);
  restrict(res32,res3, (nx/4), (ny/4), (nz/4));

  vector<double> phi3,x3,y3,z3;
  phi3.assign((((nx/4)+2)*((ny/4)+2)*((nz/4)+2)), 0.0);
  x3.assign(((nx/4)+2),0.0);
  y3.assign(((ny/4)+2),0.0);
  z3.assign(((nz/4)+2),0.0);
  double dxc3 = (1.0)/(nx/4.0), dyc3 = (1.0)/(ny/4.0), dzc3 =
      (1.0)/(nz/4.0);
  calculatexyz(x3,y3,z3,(nx/4),(ny/4),(nz/4),dxc3,dyc3,dzc3);
```

```
gS1(phi3,res3,x3,y3,z3,(nx/4),(ny/4),(nz/4),dxc3,dyc3,dzc3);

if((count == 1) || (count%20==0) )
printLevel(phi3, (nx/4), (ny/4), (nz/4), 2, count, 3);

vector<double> res43, res4;
res43.assign((((nx/4)+2)*((ny/4)+2)*((nz/4)+2)),0.0);
calResidual(phi3,res3,res43, (nx/4), (ny/4), (nz/4), dxc3, dyc3,
    dzc3);
res4.assign((((nx/8)+2)*((ny/8)+2)*((nz/8)+2)), 0.0);
restrict(res43,res4, (nx/8), (ny/8), (nz/8));
vector<double> phi4,x4,y4,z4;
phi4.assign((((nx/8)+2)*((ny/8)+2)*((nz/8)+2)), 0.0);
x4.assign(((nx/8)+2),0.0);
y4.assign(((ny/8)+2),0.0);
z4.assign(((nz/8)+2),0.0);
double dxc4 = (1.0)/(nx/8.0), dyc4 = (1.0)/(ny/8.0), dzc4 =
    (1.0)/(nz/8.0);
calculatexyz(x4,y4,z4,(nx/8),(ny/8),(nz/8),dxc4,dyc4,dzc4);
gS1(phi4,res4,x4,y4,z4,(nx/8),(ny/8),(nz/8),dxc4,dyc4,dzc4);

if((count == 1) || (count%20==0) )
printLevel(phi4, (nx/8), (ny/8), (nz/8), 1, count, 4);

/*vector<double> res54, res5;
res54.assign((((nx/8)+2)*((ny/8)+2)*((nz/8)+2)),0.0);
calResidual(phi4,res4,res54, (nx/8), (ny/8), (nz/8), dxc3, dyc3,
    dzc3);
res5.assign((((nx/16)+2)*((ny/16)+2)*((nz/16)+2)), 0.0);
restrict(res54,res5, (nx/16), (ny/16), (nz/16));
vector<double> phi5,x5,y5,z5;
phi5.assign((((nx/16)+2)*((ny/16)+2)*((nz/16)+2)), 0.0);
x5.assign(((nx/16)+2),0.0);
y5.assign(((ny/16)+2),0.0);
z5.assign(((nz/16)+2),0.0);
double dxc5 = (1.0)/(nx/16.0), dyc5 = (1.0)/(ny/16.0), dzc5 =
    (1.0)/(nz/16.0);
calculatexyz(x5,y5,z5,(nx/16),(ny/16),(nz/16),dxc5,dyc5,dzc5);
gS(phi5,res5,x5,y5,z5,(nx/16),(ny/16),(nz/16),dxc5,dyc5,dzc5);*/
//prolongation

/*vector<double> phi54;
phi54.assign((((nx/8)+2)*((ny/8)+2)*((nz/8)+2)), 0.0);
prolongate(phi5,phi54, (nx/16), (ny/16), (nz/16));
adderror(phi54, phi4, (nx/8), (ny/8), (nz/8));
gS(phi4,res4,x4,y4,z4,(nx/8),(ny/8),(nz/8),dxc4,dyc4,dzc4);*/

vector<double> phi43;
phi43.assign((((nx/4)+2)*((ny/4)+2)*((nz/4)+2)), 0.0);
prolongate(phi4,phi43, (nx/8), (ny/8), (nz/8));
```

```
    adderror(phi43, phi3, (nx/4), (ny/4), (nz/4));
    gS1(phi3,res3,x3,y3,z3,(nx/4),(ny/4),(nz/4),dxc3,dyc3,dzc3);


    if((count == 1) || (count%20==0) )
    printLevel(phi3, (nx/4), (ny/4), (nz/4), 2, count, 5);

    vector<double> phi32;
    phi32.assign((((nx/2)+2)*((ny/2)+2)*((nz/2)+2)), 0.0);
    prolongate(phi3,phi32, (nx/4), (ny/4), (nz/4));
    adderror(phi32, phi2, (nx/2), (ny/2), (nz/2));
    gS1(phi2, res2, x2, y2, z2, (nx/2), (ny/2), (nz/2), dxc2, dyc2,
        dzc2);


    if((count == 1) || (count%20==0) )
    printLevel(phi2, (nx/2), (ny/2), (nz/2), 4, count, 6);

    vector<double> phi12;
    phi12.assign(nxp2*nyp2*nzp2,0.0);
    prolongate(phi2,phi12, (nx/2), (ny/2), (nz/2));
    adderror(phi12, phi1, nx, ny, nz);
    gS(phi1, fun1, x1, y1, z1, nx, ny, nz, dxc1, dyc1, dzc1);

    if((count == 1) || (count%20==0) )
    printLevel(phi1, nx, ny, nz, 8, count, 7);

    vl2norm=l2norm(phi1,phiold,nx,ny,nz);
    //copyphi(phi1,phiold,nx,ny,nz);
    phiold = phi1;
    count = count+1;
    /*vector<double>().swap(phi2);
    vector<double>().swap(phi3);
    vector<double>().swap(res12);
    vector<double>().swap(res2);
    vector<double>().swap(phi32);
    vector<double>().swap(phi12);
    vector<double>().swap(res32);
    vector<double>().swap(res3);
    vector<double>().swap(x2);
    vector<double>().swap(y2);
    vector<double>().swap(z2);
    vector<double>().swap(x3);*/

    }while(vl2norm>0.000001);
    cout << "Total number of v-cycles:" << count;
    //printLevel(phi1, nxp2, nyp2, nzp2, 16);
    return 0;
}
```

4 Conclusion

It took 160 iterations(V-cycles) for 32x32x32 case and 40(V-cycles) iterations for 16x16x16. For the higher level of grids say 64x64x64 since it was written in C++ memory issues where there, which will be rectified in future.

4.1 Future Work

The multi grid code will be generalized for the non-uniform grid. And the memory issue will be sorted.

Thank You!