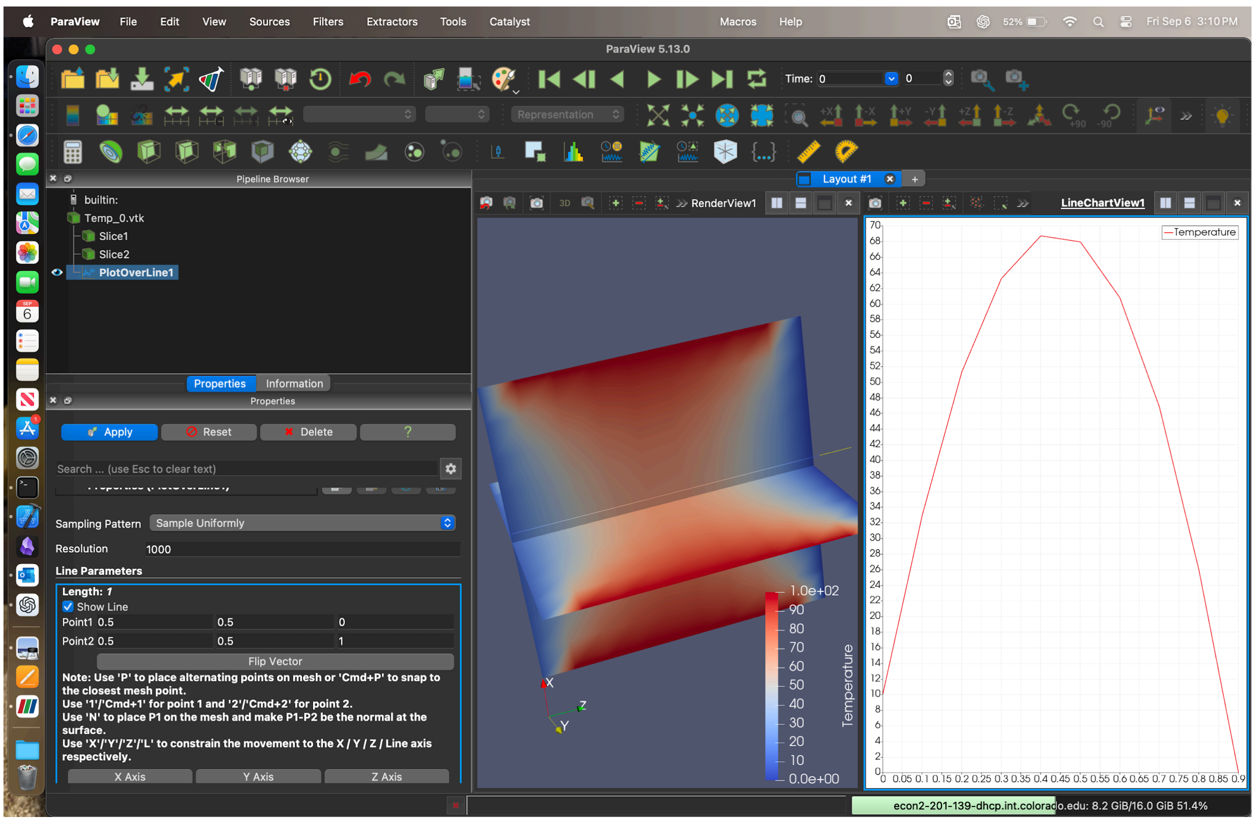
Lab 01 Report

Matthew Kind
Varun Hoskere

Table Of Contents

<u>Simulation description and results description</u>:

The main problem being solved by the completed code is the dispersal of heat through an object. The code is able to handle rectangular prisms, but in the example run it is simply a cube. Each of these complete 3D shapes is defined by and composed of cubic cells, which are where actual calculations are done. The cells are defined in terms of quantity in the x, y, and z directions, as well as the exterior (or boundary) cells having their temperatures defined before calculations are run. These differ from interior cells, where the current state of the cell needs to be calculated. Once these calculations are complete, this code outputs a vtk file that can be opened in Paraview, with these results:



The results represent the variation in temperature across the cells/nodes of the structure and this is depicted through two images - the cross section and the graph. The number of cells in x- y- and z- directions are 10 each. For the boundary conditions, each outer cell that formed the boundary was initialized with a temperature of 100 units.

Self Evaluation:

There was some confusion around the teamwork math section. We both felt most comfortable solving the systems of equations using matrices as opposed to a more brute-force method, but we were rusty on the specifics of how to carry that out for a bit. We deduced while working through the implementation of boundary conditions in FormLS that the orientation of the axes didn't matter outside of following the right-hand rule. This question took some time to answer though, given that we didn't have any convenient tools to visualize 3D problems (and hadn't used Paraview before). Once we had that understanding of how the space of the problem was laid out, we made fairly quick progress.

Appendix A: fd.cpp

```
//
==============================================================================
======
//                                                                         ||
||
//          ||                                                             fd
||
//     ||                     -------------------------------------------
||
//     ||                     F I N I T E    D I F F E R E N C E
||
//                                                                         ||
||
//     ||                   D E M O N S T R A T I O N    C O D E
||
//     ||                     -------------------------------------------
||
//                                                                         ||
||
//       ||                Developed  by:  Scott  R.  Runnels,  Ph.D.
||
//     ||                                University  of  Colorado  Boulder
||
//                                                                         ||
||
//    ||                For: CU Boulder CSCI 4576/5576 and associated labs
||
//                                                                         ||
||
//       ||                              Copyright   2024   Scott   Runnels
||
//                                                                         ||
||
//    ||                        Not for distribution or use outside of the
||
//      ||                                                  this   course.
||
```

```
//                                                                             ||
||
//
===========================================================================
======

#include "fd.h"

//  ==
//  ||
//  ||      C L A S S:   L A P L A C I A N O N G R I D
//  ||
//  ==

class LaplacianOnGrid
{

public:

  double x0, x1, y0, y1,z0, z1;
  VD x,y,z;
  int ncell_x    , ncell_y   , ncell_z, nField;
  double dx, dy,dz;
  VDD  A;
  VD   phi ;
  VD   b ;

  //  ==
  //  ||
  //  ||  Constructor:  Initialize values
  //  ||
  //  ==

   LaplacianOnGrid(double _x0    , double _x1,  double _y0, double _y1 ,
double _z0, double _z1, int _ncell_x , int _ncell_y, int _ncell_z)
  {

    x0 = _x0;    x1 = _x1;
    y0 = _y0;    y1 = _y1;
    z0 = _z0;    z1 = _z1;
```

```
  ncell_x = _ncell_x;
  ncell_y = _ncell_y;
  ncell_z = _ncell_z;
  nField  = ncell_x*ncell_y*ncell_z;
  dx      = (x1-x0)/ncell_x;
  dy      = (y1-y0)/ncell_y;
  dz      = (z1-z0)/ncell_z;

  phi.resize(nField+1);
  b.resize(nField+1);
  A.resize(nField+1);  rLOOP A[r].resize(nField+1);

}

//  ==
//  ||
//  ||  Matrix-Free Laplacian
//  ||
//  ==

void FormLS(double *bcs)
{
  rLOOP cLOOP A[r][c] = 0.;
  rLOOP       b[r]    = 0.;

  // -------------------------------------------------
  // For Lab Only
  // -------------------------------------------------
  //
  // The following line has been inserted so that
  // the code runs even though you have not
  // begun to finish it yet.  Once you have completed
  // the code, this line can be removed.  It places
  // a 1 on the diagonal, just so there are no rows
  // in the matrix that are all zeros.
  //
  rLOOP A[r][r] = 1.;
  //
  // -------------------------------------------------
```

```
    double dx2 = dx*dx;
    double dy2 = dy*dy;
    double dz2 = dz*dz;

    // Boundary Conditions

    // e-w
     kLOOP jLOOP { int r = pid(        1,         j,        k )  ; A[r][r] = 1.;  b[r] = bcs[1]; }
     kLOOP jLOOP { int r = pid( ncell_x,         j,        k )  ; A[r][r] = 1.;  b[r] = bcs[2]; }

      // In Lab: Complete the application of boundary conditions for the south-north sides (j = 0 and j = ncell_y)
    //           and for the bottom-top sides (k = 0 and k = ncell_z)

    // n-s
     kLOOP iLOOP { int r = pid(        i,         1,        k )  ; A[r][r] = 1.;  b[r] = bcs[3]; }
     kLOOP iLOOP { int r = pid(        i,         ncell_y, k )  ; A[r][r] = 1.;  b[r] = bcs[4]; }

    // t-b
     iLOOP jLOOP { int r = pid(        i,         j,        1 )  ; A[r][r] = 1.;  b[r] = bcs[5]; }
     iLOOP jLOOP { int r = pid(        i,         j, ncell_z )  ; A[r][r] = 1.;  b[r] = bcs[6]; }




    for ( int i = 2 ; i <= ncell_x - 1 ; ++i )
      for ( int j = 2 ; j <= ncell_y - 1 ; ++j )
        for ( int k = 2 ; k <= ncell_z - 1 ; ++k )
        {
            int p = pid(i,j,k);

            A[p][ p                    ] = -2./dx2 - 2./dy2 - 2./dz2;
```

```
          A[p][ pid( i-1, j  ,  k  ) ] =  1./dx2 ;

          // In Lab : Complete the formation of row p of the matrix, for
          //           interior cell at location i,j,k.


          A[p][ pid( i+1, j  ,  k  ) ] =  1./dx2 ;

          A[p][ pid( i, j-1  ,  k  ) ] =  1./dy2 ;
          A[p][ pid( i, j+1  ,  k  ) ] =  1./dy2 ;

          A[p][ pid( i, j  ,  k-1  ) ] =  1./dz2 ;
          A[p][ pid( i, j  ,  k+1  ) ] =  1./dz2 ;




       }

  }



  //  ==
  //  ||
  //  ||  Utility routines
  //  ||
  //  ==

     int  pid(int  i,int  j,int  k)  {  return  i  +  (j-1)*ncell_x  +
(k-1)*(ncell_x*ncell_y); }  // Given i-j, return point ID.  Here i-j is
the physical grid.

  #include "plotter.h"
  #include "linear_solver.h"

};




//  ==
//  ||
```

```
//   ||
//   ||  Main Program
//   ||
//   ||
//   ==

int main(int argc, char *argv[])
{


   int nPEx, nPEy, nCellx, nCelly, nCellz;

   cout << "\n";
   cout << "---------------------------------------------\n";
   cout << "\n";
   cout << " F I N I T E   D I F F E R E N C E         \n";
   cout << " D E M O   C O D E                           \n";
   cout << "\n";
   cout << "---------------------------------------------\n";
   cout << "\n";

   // Default domain size: 1 x 1 x 1 cube

   double lenx = 1.;
   double leny = 1.;
   double lenz = 1.;

   // Default BCs

   double bcs[7];
   bcs[1] = 1.;         bcs[2] = -1.;
   bcs[2] = 1.;         bcs[3] = -1.;
   bcs[4] = 1.;         bcs[4] = -1.;

   // Parse command-line options

   for (int count =  0 ; count < argc; ++count)
     {
       if ( !strcmp(argv[count],"-nCellx") ) nCellx = atoi(argv[count+1]);
       if ( !strcmp(argv[count],"-nCelly") ) nCelly = atoi(argv[count+1]);
```

```
      if ( !strcmp(argv[count],"-nCellz") ) nCellz = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-lenx"  ) ) lenx   = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-leny"  ) ) leny   = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-lenz"  ) ) lenz   = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-bce"   ) ) bcs[1] = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-bcw"   ) ) bcs[2] = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-bcs"   ) ) bcs[3] = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-bcn"   ) ) bcs[4] = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-bcb"   ) ) bcs[5] = atoi(argv[count+1]);
      if ( !strcmp(argv[count],"-bct"   ) ) bcs[6] = atoi(argv[count+1]);
    }

  // Set up LaplaciaOnGrid object

  double x0, x1;
  double y0, y1;
  double z0, z1;

  x0 =  0.;   x1 = x0 + lenx;
  y0 =  0.;   y1 = y0 + leny;
  z0 =  0.;   z1 = z0 + lenz;

  LaplacianOnGrid F(x0,x1,y0,y1,z0,z1,nCellx,nCelly,nCellz);

  // Form the linear system

  F.FormLS(bcs);

  // Solve the linear system

  F.SolveLinearSystem(500, F.b , F.phi);

  // Plot the results

  F.plot("Temp",F.phi);

  return 0;

}
```

Appendix B: Math Primer Solutions

APPENDIX B: Math Primer Solutions:

1. $$x_1 + 3x_2 + x_3 = 6$$
$$x_2 - x_3 = -3$$
$$-x_1 - 3x_2 = 12$$

$$Ax = B$$

$$\begin{bmatrix} 1 & 3 & 1 \\ 0 & 1 & -1 \\ -1 & -3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -3 \\ 12 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -3 \\ 18 \end{bmatrix}$$

$$\therefore x_3 = 18 \quad , \quad x_2 - x_3 = -3$$
$$x_2 = -3 + 18$$
$$= 15$$

$$x_1 + 3x_2 + x_3 = 6$$
$$x_1 + 45 + 18 = 6$$
$$x_1 = 6 - 63$$
$$= \underline{\underline{-57}}$$

2. $x_1 - 2x_3 = -1$

$-2x_1 + x_2 + 6x_3 = 7$

$3x_1 - 2x_2 - 5x_3 = -3$

$$\left[\begin{array}{ccc|c} 1 & 0 & -2 & -1 \\ -2 & 1 & 6 & 7 \\ 3 & -2 & -5 & -3 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 0 & -2 & -1 \\ 0 & 1 & 2 & 5 \\ 0 & -2 & 1 & 0 \end{array}\right] \begin{array}{c} \\ \\ +2R_2 \end{array} \Bigg\} \left[\begin{array}{ccc|c} 1 & 0 & -2 & -1 \\ 0 & 1 & 2 & 5 \\ 0 & 0 & 5 & 10 \end{array}\right]$$

$5x_3 = 10$

$\underline{\underline{x_3 = 2}}$

$x_2 + 2x_3 = 5$

$x_2 + 4 = 5$

$\underline{\underline{x_2 = 1}}$

$x_1 - 2x_3 = -1$

$x_1 = -1 + 4$

$\underline{\underline{x_1 = 3}}$

3. $$\begin{bmatrix} 1 & 3 & 1 \\ 0 & 1 & -1 \\ -1 & -3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -3 \\ 12 \end{bmatrix}$$

4. $$\begin{bmatrix} 1 & 0 & -2 \\ -2 & 1 & 6 \\ 3 & -2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 7 \\ -3 \end{bmatrix}$$

5. $$\begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \\ 1 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1\cdot1 + 0\cdot-1 + -2\cdot2 \\ 0\cdot1 + 1\cdot-1 + 1\cdot2 \\ 1\cdot1 + 3\cdot-1 + 1\cdot2 \end{pmatrix} = \begin{pmatrix} -3 \\ 1 \\ 0 \end{pmatrix}$$

6. $$\begin{pmatrix} 3 & -2 & 2 \\ 1 & 4 & -2 \\ 2 & -5 & 0 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \\ -1 \end{pmatrix} = \begin{pmatrix} 3\cdot2 + -2\cdot4 + 2\cdot-1 \\ 1\cdot2 + 4\cdot4 + -2\cdot-1 \\ 2\cdot2 + -5\cdot4 + 0\cdot-1 \end{pmatrix} = \begin{pmatrix} 6-8-2 \\ 2+16+2 \\ 4-20 \end{pmatrix}$$

$$= \begin{pmatrix} -4 \\ 20 \\ -16 \end{pmatrix}$$